



On the congestion control of Peer-to-Peer applications : the LEDBAT case

Claudio Testa

► To cite this version:

Claudio Testa. On the congestion control of Peer-to-Peer applications : the LEDBAT case. Other. Télécom ParisTech, 2012. English. <NNT : 2012ENST0066>. <tel-01136590>

HAL Id: tel-01136590

<https://pastel.archives-ouvertes.fr/tel-01136590>

Submitted on 27 Mar 2015

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



EDITE - ED 130

Doctorat ParisTech

T H È S E

pour obtenir le grade de docteur délivré par

TELECOM ParisTech

Spécialité «Informatique et Réseaux »

présentée et soutenue publiquement par

Claudio TESTA

le 22 Novembre 2012

**On the congestion control of Peer-to-Peer applications:
the LEDBAT case**

**Le contrôle de congestion dans les applications Pair-à-Pair :
le cas de LEDBAT**

Directeur de thèse: M. **Dario ROSSI**, Prof., Télécom ParisTech

Jury

M. **Marco AJMONE MARSAN**, Prof., Politecnico di Torino
M. **Emmanuel LOCHIN**, Dr., Institut Supérieur de l'Aéronautique et de l'Espace
Mme. **Isabelle DEMEURE** Prof., Télécom ParisTech
M. **Tijani CHAHED** Prof., Télécom SudParis
M. **Fabien MATHIEU**, Dr., INRIA
M. **Timur FRIEDMAN**, MCF, Université Pierre et Marie Curie

Rapporteur
Rapporteur
Examineur
Examineur
Examineur
Invité

**T
H
È
S
E**

TELECOM ParisTech

école de l'Institut Télécom - membre de ParisTech

Acknowledgments, Ringraziamenti

This three years have been an incredible adventure, packed with accomplishments and lessons learned, as well as some failures and hard times. But this work would not have been possible without the help of many people that I want to thank heartily.

First of all I would like to thank Dario. To you goes my biggest acknowledgement for giving me the chance to embark on this journey. Through all this time, you have been not only an invaluable supervisor, but also a constant source of inspiration and encouragement. Thank you for your guidance, for listening to my ideas, for your contagious optimism. And also for giving me the chance to brush up my baby-sitting skills!

I am grateful to all the people I had the chance to work with, for their contributions and expertise: Luca, Giovanna, Arnaud, Ashwin, Timur and Antonio.

I would like to thank also the members of the jury, for attending my dissertation, and in particular the reviewers of the manuscript, for their suggestions and comments.

A special thought goes to all the colleagues of Télécom ParisTech/LINCS and friends I met in Paris, for the memorable moments we shared. For the everyday life in lab, but also the pizzas, the coffees, the nights out. Thank you Silvio, Paolo, Eugenio, Massimo, Davide C., Antonio, Stefano, Aruna, Mattia, Giuseppe, Matteo, Marianna, Erika, Jessica, Nico, Rosa, Davide T., Sharon, Chiara, Raffaele, Amy, Sameh, Ahlem, Ghida, Rim, Sébastien, Xavier, Thomas, Maciej, Anand, Mathilde, Walter, Michele.

The mates of the Issy Triathlon club, for the after-work diversion, and for having taught me the importance of commitment and perseverance, esteemed values not only in sport.

Ringrazio i miei amici in Italia, che sono sempre pronti ad accogliermi calorosamente ad ogni ritorno a casa, e con i quali i legami son sempre forti, nonostante i chilometri che ci separano.

I apologize for who I might have missed, but be assured that I do care about you too. We might have crossed our path just for a little bit, but I will remember you for the years to come.

Finally, but not less importantly, I would like to express my gratitude to my family. Grazie per avermi lasciato partire tre anni fa, per avermi sempre supportato e incoraggiato, per essere stati presenti anche nella lontananza. Grazie.

Abstract

In the last years, Internet delays are considerably growing, causing a performance deterioration of interactive applications. This phenomenon is getting worse with the increasing popularity of bandwidth-intensive applications, as video streaming, remote backup and distributed content dissemination systems. The cause of these delays has been identified with the excess buffering inside the network, called “bufferbloat”. Research efforts in this direction head toward active queue management techniques and end-to-end congestion control. However, an ultimate solution to this problem is yet to come.

In this context, we investigated LEDBAT, a low-priority delay-based transport protocol introduced by BitTorrent. This protocol is specifically designed to transfer large amount of data without affecting the delay experienced by other applications or users.

First, we analysed transport-level performance of LEDBAT by means of experimental measurement, simulation and analytical model. Specifically, we evaluated LEDBAT as it is, comparing its performance to standard TCP or to other lower-than best effort protocols. We then identified a later-comer advantage affecting the original proposal. We tackled this unfairness issue proposing fLEDBAT, which re-introduces intra-protocol fairness maintaining the original LEDBAT objectives.

As a second step, we studied the impact of the LEDBAT protocol on BitTorrent performance. More in details, through simulations and real network experiments, we analysed how BitTorrent impacts on the buffer occupancy of the access node. BitTorrent performance was evaluated in terms of completion time, which can be considered as the cardinal metric to assess the user quality of experience. In both scenarios, results showed that LEDBAT decreases the completion time with respect to standard TCP. This is an unexpected phenomenon, to which we find a preliminary, though not conclusive, explanation. Moreover, LEDBAT significantly reduces the buffer occupancy, that translates in lower delays experienced by competing interactive applications.

Résumé

Durant ces dernières années, les délais de transmission sur Internet ont augmenté de manière considérable, causant une détérioration de performances des applications interactives. Ce phénomène s'aggrave avec la popularité croissante des applications gourmandes en bande passante, telles que le streaming vidéo, les systèmes distants de sauvegarde de données et de diffusion de contenus.

La cause de ces augmentations de délais est à imputer à l'excès de mémoire tampon à l'intérieur du réseau, appelé "bufferbloat". Les efforts de recherche dans cette direction vont vers des techniques de gestion des files d'attente actives et des techniques de contrôle de congestion de bout-à-bout. Cependant, une solution définitive à ce problème reste encore à trouver.

Dans ce contexte, nous avons examiné LEDBAT, un protocole introduit par BitTorrent qui se base sur le délai au niveau transport. Ce protocole est spécifiquement conçu pour transférer de grandes quantités de données sans affecter les délais de transmission expérimentés par d'autres applications ou utilisateurs.

Nous avons tout d'abord analysé la performance de niveau de transport de LEDBAT au moyen de mesures expérimentales, de simulations et de modèles analytiques. Plus précisément, nous avons évalué LEDBAT en comparant ses performances au standard TCP ou à d'autre protocoles de priorité faible. Nous avons ensuite identifié un problème d'iniquité, où le dernier flux arrivant obtient de meilleures performances que les flux déjà présents. Nous avons abordé cette question en proposant fLEDBAT, qui ré-introduit l'équité intra-protocole en maintenant les objectifs d'origine de LEDBAT.

Dans un deuxième temps, nous avons étudié l'impact du protocole LEDBAT sur la performance de BitTorrent. Plus en détail nous avons analysé, par des simulations et des expérimentations sur réseaux réelles, les effets de LEDBAT sur le remplissage des tampons des nœuds d'accès. Les performances de BitTorrent ont été évaluées en termes de temps d'exécution, ce qui peut être considéré comme la métrique cardinale permettant d'évaluer la qualité de l'expérience utilisateur. Dans les deux cas, les résultats ont montré que LEDBAT diminue le temps de traitement par rapport au standard TCP. Ceci est un phénomène inattendu, auquel nous trouvons une première explication, quoique non concluante. De plus, LEDBAT réduit de manière significative l'utilisation de tampons, ce qui se traduit par une baisse des délais subis par les applications interactives concurrentes.

Synthèse en Français

Introduction

Comme récemment indiqué dans [30], les délais de l'Internet sont autant présents que problématiques. Avec l'utilisation croissante de services gourmands de bande passante, les performances des applications interactives ont rencontré une détérioration nette. En conséquence, les utilisateurs finaux souffrent de retards, importants et variables, qui entravent le bon fonctionnement des applications sensibles à la latence (par exemple la VoIP, la navigation Web, les jeux en ligne, l'administration à distance) et peuvent empêcher le bon fonctionnement de protocoles de base (par exemple déclencher les temporisations de DNS).

La cause première de ces retards est l'utilisation excessive des tampons à l'intérieur d'un réseau. Ce phénomène est appelé "bufferbloat". Bien que cette problématique soit bien connue, depuis sa première étude dans [31], ces dernières années ont vu un sur-dimensionnement des tampons, résultat du coût réduit de la mémoire, qui a aggravé ce problème. Cette augmentation de la capacité des tampons dans les périphériques réseaux a l'avantage de réduire les pertes de paquets, mais empêche le bon fonctionnement de TCP, qui compte sur les pertes de paquets pour détecter la congestion sur le chemin.

Une augmentation excessive de la mémoire tampon a pour effet une accumulation des paquets dans les files d'attente quand ils passent d'une portion rapide du réseau à une portion lente. Ce phénomène se produit généralement sur la liaison montante des abonnés vers le cœur de réseau, où les débits importants fournis par les réseaux locaux convergent vers le faible débit de la connexion ADSL à Internet. Ces files d'attente ont inévitablement besoin de temps pour se vider et cela se traduit par une réponse ralentie et des retards importants, qui peuvent atteindre jusqu'à quelques seconds [16, 58].

Afin de résoudre ce problème, plusieurs propositions ont été faites et peuvent être classées dans deux catégories indépendantes. D'une part, certains chercheurs [30] recommandent des techniques de gestion active de file d'attente (AQM), comme solution ultime pour réduire les délais d'attente. D'un autre côté, nous trouvons des propositions se dirigeant vers l'ingénierie de protocoles de contrôle de congestion (CC) permettant de remplacer TCP. En particulier, pour les services de transfert à faible priorité, on trouve plusieurs propositions dans la littérature qui ciblent un comportement *Lower-than Best Effort* (LBE), tels que TCP-LP [60], NICE [81], 4CP [63], Microsoft BITS [12], et plus récemment, LEDBAT [79].

LEDBAT, nommé d'après *Low Extra Delay Background Transport*, est un protocole de contrôle de congestion pour la transmission de données au-dessus d'UDP, introduit en fin 2008 par BitTorrent [69]. L'intuition derrière ce protocole est nouvelle dans le paysage des algorithmes de contrôle de congestion. De manière raisonnable, LEDBAT émet l'hypothèse que le goulot d'étranglement est plus probablement situé à l'accès du réseau (par exemple, sur la ligne ADSL de l'utilisateur). Cela signifie que la congestion est généralement *auto-induite* par des sessions simultanées de trafic générées par l'utilisateur même (par exemple, des transferts BitTorrent en parallèle avec des appels Skype et de la navigation Web). En conséquence, en raison des tampons, de tailles importantes, implémentés au niveau des nœuds d'accès, la congestion provoque une augmentation des délais, ce qui a aggravé le phénomène bufferbloat.

Ce nouveau protocole est conçu pour éviter d'introduire un délai excessif dû au bufferbloat et cible des transferts (i) efficaces mais (ii) à faible priorité. Lorsque les flux LEDBAT obtiennent l'usage exclusif des ressources, ils peuvent exploiter pleinement la capacité disponible. Néanmoins, lorsque les transferts –telles que la VoIP, les jeux en ligne, le Web ou d'autres flux TCP– sont en cours, LEDBAT se bride afin d'éviter de nuire les performances du trafic interactif.

Cet objectif est atteint en faisant réagir LEDBAT avant que TCP ne perçoive la congestion et ne réduise le taux de transmission, permettant ainsi de ne pas nuire au transferts TCP en cours. Alors que TCP détecte une congestion avec le pertes de paquets, LEDBAT déduit la congestion grâce à l'augmentation des délais de transmission, donc avant que les pertes se produisent. Ainsi, une autre contribution importante de LEDBAT est qu'il constitue un soulagement pour les opérateurs, car ils n'ont plus besoin de limiter le trafic P2P, maintenant géré de manière plus douce [4].

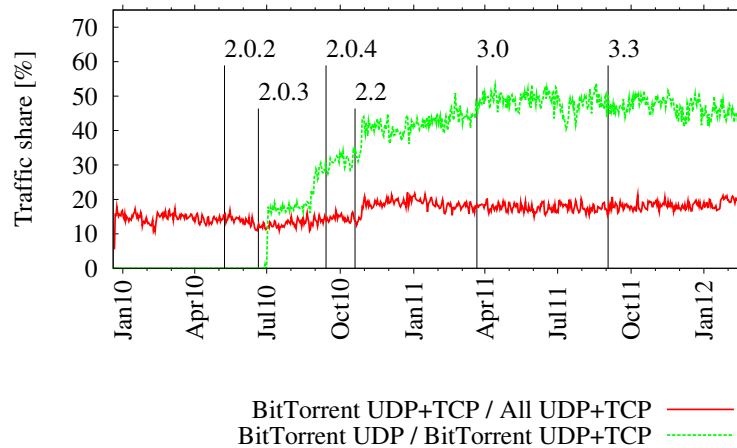


Figure 1: Proportion du trafic BitTorrent sur TCP et LEDBAT dans un réseau réel.

Afin de souligner l'importance de ce protocole, nous montrons sur Fig. 1 des mesures dans le réseau d'un opérateur réel, où nous pouvons voir sa diffusion actuelle. À partir du mois de Mars 2010, au moment où LEDBAT est devenu le protocole par défaut dans la version officielle du logi-

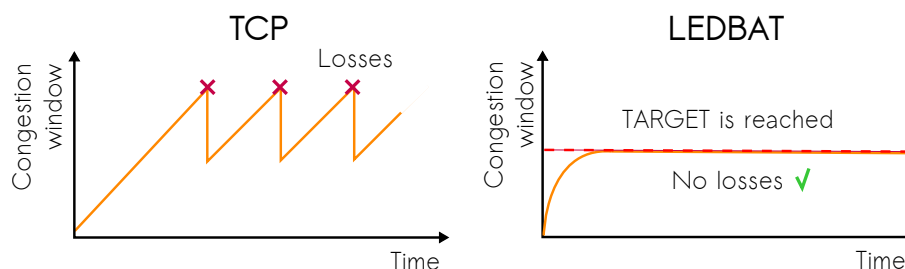


Figure 2: Comparaison de l'évolution dans le temps de la fenêtre de congestion de TCP et LEDBAT.

ciel BitTorrent nous voyons que le pourcentage de trafic BitTorrent UDP augmente énormément, jusqu'à représenter à peu près 50% de la totalité des données échangées sur le réseau. Nous voyons aussi que la pourcentage de trafic BitTorrent sur le réseau montre une légère inflexion, mais cette tendance est en train de changer, comme montré par des études plus récentes [42]. Cette grande diffusion du protocole met en évidence l'importance de ce travail, qui vise à étudier les mécanismes internes de LEDBAT, à vérifier la validité de ses objectifs et à éclaircir les avantages et les inconvénients de ses choix de conception.

LEDBAT: spécification et évolution

En fin 2009, un message sur le forum développeur BitTorrent, annonçant la sortie de la nouvelle version 1.9-alpha-13485 [69] de l'application μ Torrent a suscité beaucoup d'intérêt ainsi que tout un buzz peu motivé [13]. Non seulement le client BitTorrent officiel devenait *closed-source*, mais il introduisait avant tout une nouvelle couche au niveau applicatif pour le transfert de données, nommé *micro Transport Protocol* (uTP). Pourtant, le buzz n'a pas été lancé sur une base technique solide : en effet, l'annonce originelle [69] indiquait clairement que l'objectif de conception de ce nouveau protocole était d'éviter de tuer les connexions réseaux concurrentes –même sans avoir fixé de limite de débit sur l'application BitTorrent. Le fonctionnement interne de ce nouveau protocole est en cours de discussion au sein de BitTorrent Enhancement BEP-29 [71] sous le nom de uTP, ainsi qu'au sein de l'IETF *Low Extra Delay Background Transport* (LEDBAT) groupe [79].

LEDBAT est conçu pour détecter l'apparition de la congestion sur un chemin avant que TCP ne le fasse. Son algorithme de contrôle de congestion est basé sur l'estimation du délai de transmission sur l'aller : le délai d'attente est estimé comme étant la différence entre le délai instantané et un délai de base, celui-ci étant le plus petit délai sur la précédente observation. Chaque fois que l'expéditeur détecte un délai en augmentation, il déduit que la file d'attente est en train de grandir, et il réagit en diminuant son taux d'envoi à travers un régulateur linéaire. Ainsi, LEDBAT réagit plus tôt que le protocole TCP, qui attends une perte de paquet pour détecter la congestion. Dans le même temps, LEDBAT essaie d'ajouter seulement une quantité fixe de délai d'attente (paramètre `TARGET` de l'algorithme) dans le tampon, comme est montré dans la comparaison vi-

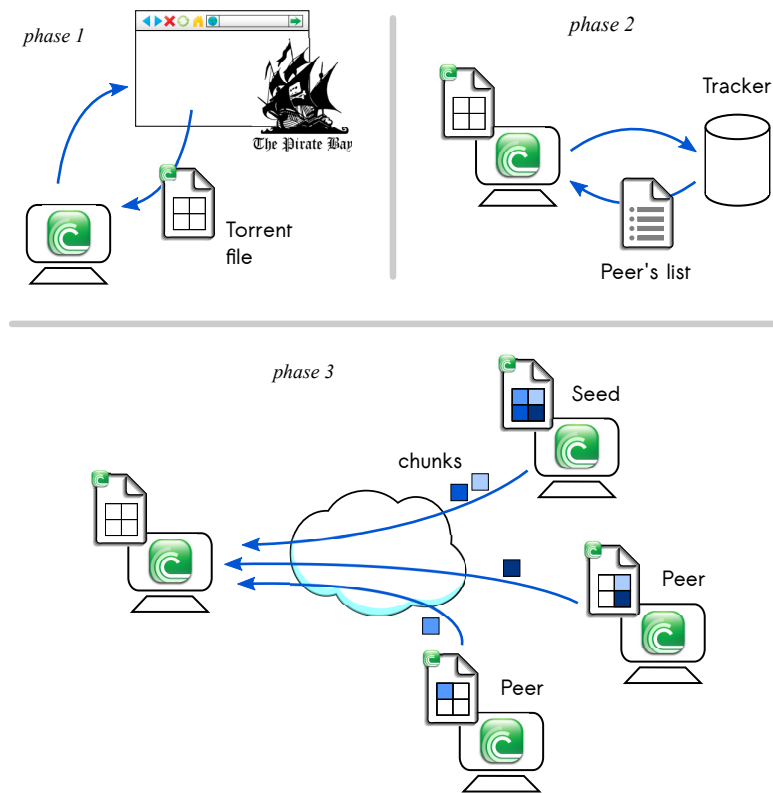


Figure 3: Diagramme de fonctionnement de BitTorrent en 3 phases.

suelle en Fig. 2. Alors que les principes de conception de LEDBAT sont solides, certains points techniques ont été soulevés par la communauté scientifique qui participe au groupe de travail sur LEDBAT et sont actuellement en phase de discussion. Une question légitime est de savoir si l'ajout de LEDBAT au monde déjà bien peuplé des algorithmes de contrôle de congestion sur Internet est vraiment nécessaire et motivé. Les réticents à LEDBAT suggèrent en effet de considérer les algorithmes déjà existant, et donc plus stables et mieux compris, pour le transport de faible priorité, comme NICE [81] ou TCP-LP [60]. Ces observations, associées à la transition vers un code-source fermé, ont motivé la nécessité d'études indépendantes, de sorte que les allégations relatives, par exemple, à l'équité sur le partage des ressources et à l'efficacité de ce nouveau protocole, peuvent être confirmée par la communauté scientifique. Ceci a motivé le travail effectué dans cette thèse.

Vue d'ensemble sur BitTorrent

BitTorrent représente aujourd'hui l'un des exemples les plus réussis de service P2P. Il a lentement eu raison d'un service clé traditionnellement offert par le paradigme client-serveur : la distribution de contenu. La force de ce protocole réside dans sa capacité de passer à l'échelle, sa capacité à tra-

vailler sur des réseaux non fiables et hétérogènes, et ses mécanismes d'incitation, qui encouragent l'échange de données entre les nœuds.

Contrairement aux applications traditionnelles de partage de fichiers, basées sur le modèle client-serveur, l'objectif principal de BitTorrent est de développer un système efficace qui implique un grand nombre d'hôtes, appelés *peers*, qui peuvent télécharger la ressource et l'envoyer aux autres peers en même temps. Pour accélérer la distribution de contenus et augmenter le nombre de peers impliqués, la ressource d'origine est divisée en petits fragments, ou *chunks*, puis en blocs plus petits (*blocks*). Le fichier torrent est le fichier de métadonnées contenant l'index de tous les morceaux d'une ressource, ainsi que l'empreinte digitale utilisée pour vérifier l'intégrité de chaque chunk. Pour commencer le téléchargement, un peer doit récupérer le fichier torrent à partir d'un site d'hébergement Web (*phase 1*). En utilisant les informations contenues dans le fichier torrent, le peer peut communiquer avec un *tracker*, un serveur centralisé qui fournit une première liste de peers voisins intéressés à la même ressource (*phase 2*). Cette première liste est ensuite périodiquement mise à jour par le tracker lui-même et intégré avec l'échange des listes des peers voisins. Enfin, le peer peut commencer à demander les chunks à d'autres peers impliqués dans cet échange, appelé *swarm* (*phase 3*). À la fin, les chunks sont ré-assemblés à la destination, une fois qu'ils ont tous été téléchargés correctement. Une illustration simplifiée de ces 3 phases est présentée dans Fig. 3.

Contributions

Les objectifs que cette thèse vise à atteindre sont les suivants :

- Apporter la lumière sur les mécanismes internes de LEDBAT et ses choix de conception.
- Comprendre le comportement de l'application BitTorrent dans un environnement contrôlé et dans une large gamme de scénarios et paramètres de configuration.
- Quantifier le niveau de faible priorité de LEDBAT, par rapport aux autres protocoles de contrôle de congestion, et l'adaptabilité de son comportement, au travers des principaux paramètres de fonctionnement.
- Prouver et corriger les failles du protocole, mises en évidence par le modèle analytique, les simulations et les expériences de mesure.
- Quantifier l'impact de LEDBAT sur les performances des flux TCP, aussi bien dans des scénarios de simulation que dans des cas réels.
- Évaluer l'impact de LEDBAT sur l'application BitTorrent à travers des simulations et expériences en réseaux réels.

La première partie de cette thèse est consacrée à l'analyse du protocole LEDBAT du point de vue du contrôle de congestion. Dans ce but, différentes méthodologies sont exploitées, comme (i) un testbed expérimental, (ii) des programmes de simulation, et (iii) un modèle analytique. Afin

de comparer les performances du nouveau protocole par rapport au standard TCP ou aux autres solutions LBE, différentes métriques orientées sur les flux sont utilisées –par exemple l'équité intra- et inter-protocole, l'efficacité d'exploitation du réseau, l'occupation du tampon, pour ne citer que quelques-unes.

La deuxième partie de la thèse est axée sur LEDBAT comme un acteur majeur dans l'écosystème P2P. Pour cette raison nous avons étudié son influence sur la performance de BitTorrent, soit avec des simulations, soit avec des expériences sur des réseaux réels.

Évaluation de performance au niveau flux

Étude de mesure

Notre première étude du protocole LEDBAT, présenté dans Chap. 2, est basé sur des mesures, d'abord effectuées sur un testbed actif et en suite sur Internet. Notez que, au moment où cette étude a été faite, aucune description publique de LEDBAT n'était disponible. En fait, les développeurs de BitTorrent avait annoncé le nouveau protocole basé sur UDP, mais ils l'avaient seulement implémenté dans la version bêta du logiciel, sans publier une description du protocole même. Pour cette raison, un approche *black-box* sur l'application BitTorrent était la seule méthodologie possible pour étudier ce protocole.

Depuis sa première version publiée en fin 2008, le client est passé par de nombreuses versions qui on permis le perfectionnement de l'implémentation adoptée. Afin d'apprécier les modifications et les améliorations introduites au fur et à mesure par les développeurs, nous avons essayé de suivre son évolution, en répétant les mêmes expériences pour différentes versions successives du logiciel. Comme précédemment mis en avant nous avons utilisé un testbed actif pour effectuer nos expériences : nous connectons le client qui tourne sur différentes machines sur un router Linux, sur laquelle nous utilisons `netem` afin d'émuler des conditions de réseau (délai et débit) arbitraires. De cette façon, nous pouvons facilement étudier le comportement du protocole dans différentes situations, et ainsi observer comment il s'adapte à des conditions difficiles fluctuant beaucoup.

Flow unique

Dans notre première expérience nous verrons comment le protocole utilise la bande passante disponible. En Fig. 4-(a), nous montrons le débit de différentes versions du protocole (α_1 , α_2 , β_1 qui correspondent à deux alpha versions de LEDBAT et à la première stable beta) quand elles sont toutes seules sur un lien à 10 Mbps. À noter que chaque courbe correspond à une expérience différente, mais nous les avons superposée afin d'obtenir une meilleure comparaison. Pour référence, nous montrons aussi le débit atteint par TCP classique, dans deux implémentations, celle de Linux et celle de MS Windows. Nous observons tout suite un dysfonctionnement dans la première version α_1 , qui n'est pas capable d'avoir un débit constant. L'erreur a été corrigée dans α_2 , mais le protocole n'est pas encore capable de profiter de toute la bande disponible, probablement à cause de mauvaises valeurs de paramétrage. Cela a été fixé dans β_1 qui ne souffre plus de ce

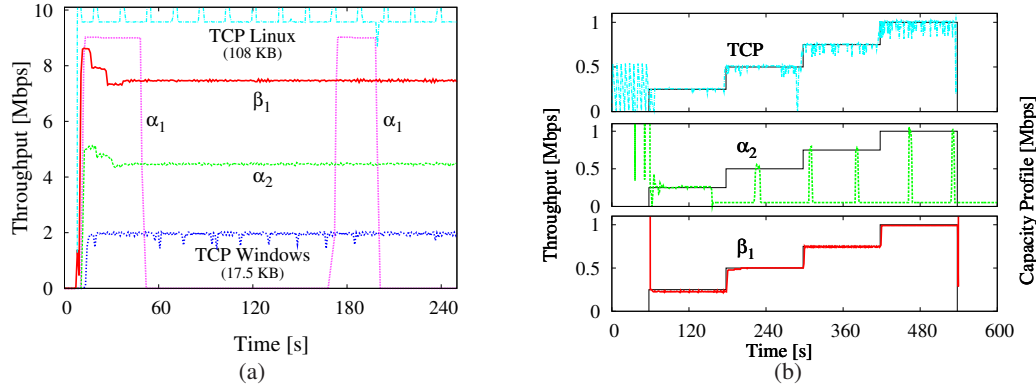


Figure 4: Débit pour différents versions du protocole (a) sans ou (b) avec des limitation de capacité sur le lien.

problème. Si on compare avec les deux TCP, nous voyons que TCP Windows n'arrive pas non plus à bien exploiter la capacité du lien, à cause d'une valeur maximale de la fenêtre de réception maximale trop petit (nous précisons que cette observation est valable pour Windows XP Home édition; dans les versions plus récentes une valeur plus adaptée aux réseaux moderne permet de profiter de toute la capacité), alors que la version Linux utilise correctement toute la capacité disponible.

Sur le graphique Fig. 4-(b), nous changeons la capacité du lien en augmentant à des intervalles réguliers la bande disponible de 0 à 1 Mbps avec des pas successives de 250 Kbps. Encore un fois, nous pouvons remarquer le travail des développeurs : si la première α_1 n'arrive pas à s'adapter aux conditions de capacité variable, β_1 est tout à fait très efficace à égaler la capacité disponible, comme la plus mature implémentation TCP. De plus, étant basé sur le délai, nous pouvons remarquer aussi une meilleure stabilité, par rapport aux classiques oscillations de TCP autour du débit moyen. Nous pouvons remarquer le même phénomène dans la Fig. 5-(a), où nous observons le débit atteint par un flux TCP et un flux LEDBAT sur un lien ADSL sur Internet. Les deux courbes correspondent de nouveau à deux expériences différentes et sont superposées pour mieux les comparer. La plus importante observation est que le débit de LEDBAT est beaucoup plus stable, alors que TCP présente des oscillations évidentes, qui sont dûs à la contrôle de congestion basé sur les pertes.

Dans la deuxième partie de chapitre Chap. 2, nous conduisons une étude similaire mais en changeant le délai sur le lien (qui dans les précédentes expériences était fixé sur 50 ms). Notre résultat plus important est que le délai sur le parcours de retour, qui ne devrait avoir aucune influence sur LEDBAT, vu qu'il mesure seulement le délai d'aller, a en fait un impact non négligeable car il ajoute du retard dans la boucle du contrôleur LEDBAT résultant en un débit plus instable.

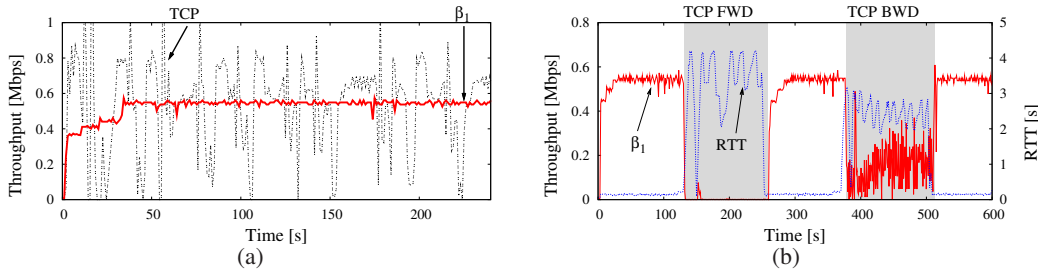


Figure 5: Expériences sur réseau Internet : (a) débit moyen et (b) comportement de LEDBAT avec de trafic d'interférence.

Flux multiples sur Internet

Finalement, sur Fig. 5-(b) nous étudions le comportement de LEDBAT lorsqu'il est concurrent à un flux TCP sur le même goulot d'étranglement. L'expérience a aussi été faite sur un lien ADSL commercial sur Internet. Nous montrons le débit atteint par le flux LEDBAT, ligne rouge, et le RTT mesuré en même temps avec ping en bleu. Dans les deux périodes de temps démarqué par la zone grise, nous avons lancé un flux TCP concurrent, le première dans la même direction du flux LEDBAT et le deuxième dans le sens inverse. Les courbes montrent que LEDBAT profite correctement de toute la bande disponible quand il est seul sur le lien et qu'il réduit le débit correctement pour laisser la priorité au flux TCP dans le même sens. Pourtant, quand il y a le flux dans le sens inverse, le comportement est différent de ce que l'on attendait : LEDBAT n'est pas insensible à cela, et l'effet d'un retard additionnel ainsi que le trafic de ACK sur le parcours de retour influe fortement sur le débit atteint.

Étude de simulation

Dans le chapitre Chap. 3, nous procédons à une étude en profondeur de LEDBAT, après que ses spécifications aient été publiées à la fin de 2009 avec un Draft IETF [79]. Ensuite, nous nous concentrons sur une comparaison des performances de la nouvelle proposition avec des autres protocoles LBE, tels que TCP-LP [60] et NICE [81], afin de quantifier les niveaux d'agressivité fournis par ces protocoles. En outre, nous procédons à une analyse de sensibilité attentive des performances LEDBAT, en réglant ses paramètres principaux à la fois dans des scénarios inter-protocole (contre TCP) et intra-protocole (contre LEDBAT même).

Nous décrivons d'abord l'algorithme LEDBAT tel qu'il est présenté dans le Draft, en se concentrant sur ses choix de conception, afin de définir les buts et objectifs qu'il vise à atteindre. Pour faciliter cette tâche, nous avons repris le pseudo-code dans la Fig. 6, dans sa forme la plus simple, ce qui nous permet de mieux le comprendre.

La partie de l'algorithme coté récepteur est plutôt simple. En fait, elle se limite à calculer le délai des paquets reçus comme différence entre le timestamp du paquet et sa propre horloge interne. Ce délai est par la suite envoyé à l'émetteur, où se passe la plus partie des opérations de

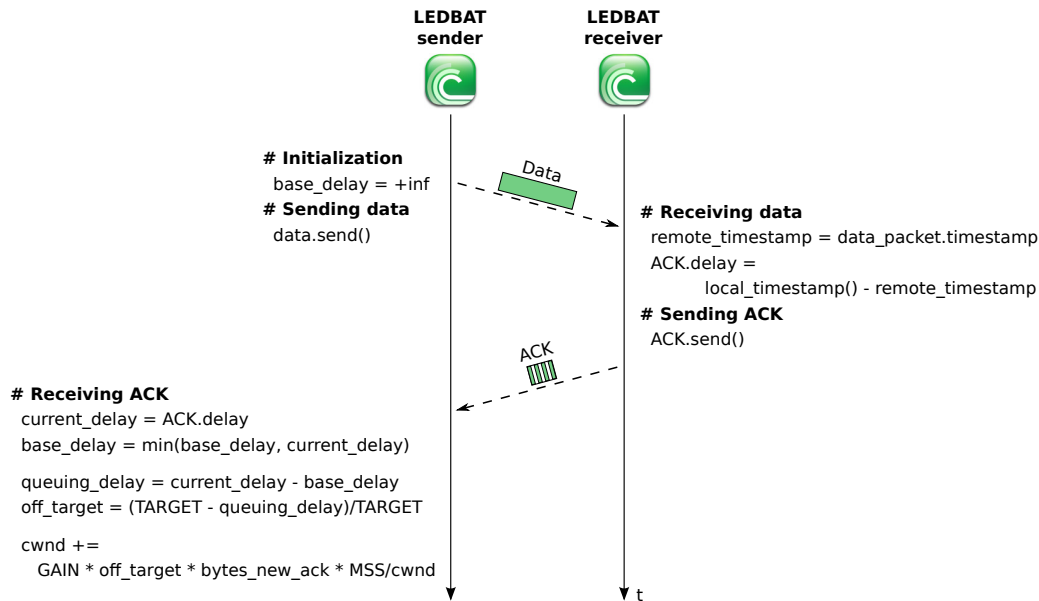


Figure 6: Pseudocode de LEDBAT coté émetteur et receveur.

l'algorithme. L'émetteur garde un historique des délais mesurés : le minimum observé constitue le `base_delay` qui veut être une estimation de la partie constante du délai, autrement dit le *délai de propagation*, qu'on observe quand la file d'attente est vide. En conséquence, la différence entre le délai instantané et le délai de propagation correspond au délai dû à la file d'attente. Le but de LEDBAT est d'introduire un délai additionnel sur le parcours égale à `TARGET`, qui était originalement égal à 25 ms et a récemment été fixé à 100 ms. Ensuite, la fenêtre de congestion est modulée par un contrôleur linéaire en fonction de la distance du délai mesuré depuis le `TARGET`. Le dernier paramètre de l'algorithme est donc le coefficient multiplicatif du contrôleur, nommé `GAIN`, qui n'était pas spécifié dans la version originelle du Draft. Nous allons choisir $1/\text{TARGET}$ comme valeur de `GAIN`, selon les directives qui spécifient que le contrôleur ne puisse pas grimper plus vite qu'un TCP standard. Ce comportement contribue au principal objectif du protocole qui était de fournir un service à faible priorité. Pour atteindre cela, il est aussi requis que le protocole détecte la congestion avant TCP et qu'il réagisse plus vite : comme LEDBAT commence à diminuer son propre débit dès qu'il mesure un délai plus grand que `TARGET`, sa réaction précède celle de TCP standard.

Vue d'ensemble sur LEDBAT

Nous avons implémenté cette algorithme dans le simulateur de réseau au niveau paquet `ns2`. L'algorithme à été implémenté comme un nouveau contrôle de congestion pour TCP, en utilisant l'option `timestamp`, disponible dans TCP pour calculer les délai. Comme le simulateur utilise

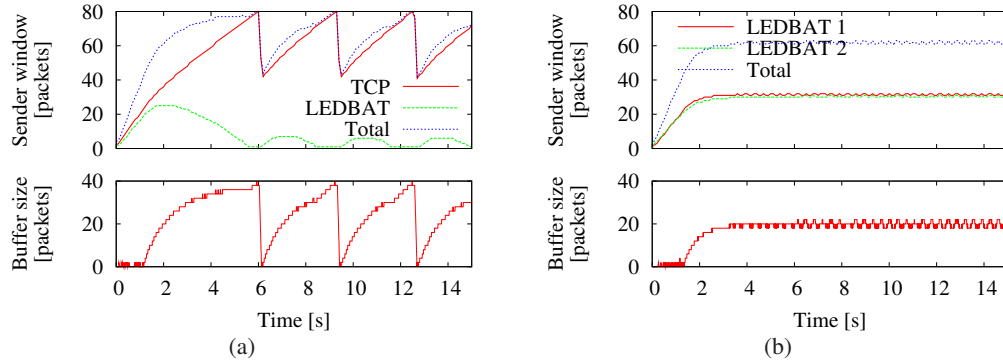


Figure 7: Évolution temporelle de la fenêtre de congestion de l'émetteur (en haut) et de la taille de la file d'attente (en bas) pour les protocoles (a) TCP-LEDBAT et (b) LEDBAT-LEDBAT.

le même système modulaire que celui utilisé dans le noyau de Linux, notre code, disponible au téléchargement ici [7] peut être aussi utilisé dans un système d'exploitation réel.

Avec cet outil, nous avons pu étudier le comportement du protocole dans des scénarios très simples, afin de déterminer d'abord s'il atteint ses objectifs. La plus importante caractéristique de LEDBAT veut être sa priorité basse : nous allons donc étudier le comportement d'un flux LEDBAT et un flux TCP qui partagent le même lien. L'évolution temporelle des fenêtres de congestion et du dimensionnement de la file d'attente est représenté dans la Fig. 7-(a). Nous voyons que le flux TCP n'est pas du tout influencé par la présence de LEDBAT sur le même lien: nous pouvons observer en fait qu'à chaque fois que la file d'attente dépasse les 20 paquets, le flux à basse priorité diminue son débit pour laisser l'espace flux TCP, plus agressif, qui garde son comportement habituel à dent de scie. Pourtant, LEDBAT est capable d'utiliser la bande laissée libre par le flux TCP, ce qui augmente l'utilisation total du lien.

Dans la Fig. 7-(b) nous observons le comportement de 2 flux LEDBAT qui partagent le même lien. Nous voyons que, comme ils ont commencé en même temps, ils ont aussi mesuré le même délai de propagation. Il estiment donc correctement la dimension de la file d'attente et arrivent facilement à partager de façon équitable la capacité. En outre, la taille de la file d'attente reste très stable, 20 paquets, ce qui correspondent aux 25 ms de TARGET délai spécifiés dans le Draft dans ses premières versions.

Analyse de sensibilité sur target et gain

Nous commençons notre analyse de sensibilité en considérant deux flux démarrant simultanément, un standard TCP Reno et un LEDBAT, et nous modifions à la fois les valeurs des paramètres TARGET et GAIN. Notez que le Draft IETF ne spécifie aucune valeur pour le paramètre de GAIN, mais spécifie une valeur obligatoire pour le paramètre TARGET 25 ms (augmenté à 100 ms dans

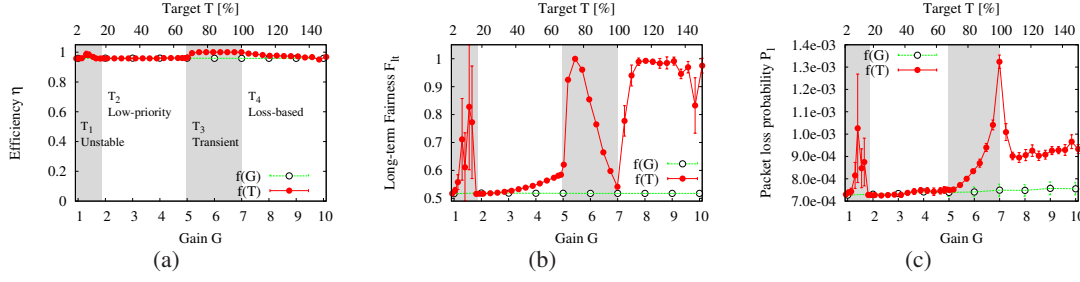


Figure 8: LEDBAT vs TCP Reno: analyse de sensibilité inter-protocole, pour différents valeurs des paramètres target T et gain G .

les dernières versions). Ce choix de target est en quelque mesure arbitraire, basé sur des observations expérimentales, dont les résultats ne sont pas publiés. En tant que tel, le target est souvent appelé “nombre magique”, avec un sens péjoratif, dans le groupe de travail sur LEDBAT [6]. Par conséquent, nous pensons qu’une exploration approfondie de l’impact de ces paramètres est nécessaire, et nous la réalisons par simulation.

Afin de comparer les performances, nous présentons ici trois métriques, qui concernent les performances du point de vue du réseau (par exemple, l’efficacité η) ou de l’utilisateur final (par exemple, l’équité F et le taux de perte des paquets P_l). En détail, η exprime l’utilisation du lien comme le rapport entre la somme des valeurs de débits x_i atteints par l’ensemble des flux sur la capacité disponible : $\eta = \frac{\sum_i x_i}{C}$. F décrit le partage des ressources, dans le cas où nous avons N flux qui tentent de transmettre en même temps sur le canal : $F = \frac{(\sum_{i=1}^N x_i)^2}{N \cdot \sum_{i=1}^N x_i^2}$. Enfin, comme dernier indicateur de performances pour l’utilisateur, nous considérons la probabilité de perte des paquets P_l , qui est calculé comme le rapport des paquets perdus par rapport au nombre total de paquets envoyés sur le lien.

Dans Fig. 8 nous montrons les résultats de simulation pour chacune des métriques η , F et P_l . Dans chaque figure, nous présentons deux courbes, à savoir $f(G)$ et $f(T)$. La fonction $f(G)$ décrit comment $f(\cdot)$ varie en fonction du gain $G \in [1, 10]$, avec un target fixé à 25 ms. La fonction $f(T)$ décrit comment $f(\cdot)$ varie en fonction du target $T \in [2, 150]\%$ par rapport à la valeur de référence, lorsque le gain est fixé à 1.

Les résultats montrent que, pour toutes les métriques, la courbe de la fonction $f(G)$ est à peu près stable. Cela signifie que le paramètre GAIN a très peu d’influence sur le comportement du protocole LEDBAT. Cela est dû au fait que LEDBAT est conçu pour céder rapidement le passage à TCP, quel que soit la valeur de G .

Par conséquent, nous limitons notre attention à l’impact du paramètre TARGET, en analysant le comportement de la courbe $f(T)$. Sur Fig. 8-(a), nous pouvons voir que l’efficacité η est peu influencée par la variation du T et reste toujours proche de la capacité totale du lien. Compte tenu de l’équité indiquée dans la figure Fig. 8-(b), on peut identifier quatre régions de travail. Lorsque

le target est très bas $T_1 \in [2, 20]\%$ le protocole LEDBAT n'est pas en mesure d'atteindre le délai souhaité, ce qui conduit à un comportement instable. Dans une seconde région $T_2 \in [20, 65]\%$, nous voyons que LEDBAT cède complètement le passage à TCP Reno, travaillant en mode à faible priorité, conformément à ses objectifs (l'équité est proche du minimum $F \simeq 0.5$). Dans une troisième région $T_3 \in [65, 100]\%$, LEDBAT commence à éroder de la bande passante au flux TCP Reno. Cela provoque des pertes dans le flux TCP Reno, qui réduit progressivement son débit. Par conséquent, l'équité se rapproche du maximum en cas de partage égal ($F \simeq 1$), mais une croissance excessive du target résuit le débit alloué à TCP Reno. Enfin, dans la quatrième région $T_4 > 100\%$, le target est supérieur à la taille du tampon : dans ce cas, LEDBAT retombe dans un comportement basé sur les pertes des paquets, comme TCP Reno. La probabilité de perte de paquets P_l , montrée en figure Fig. 8-(c), confirme ce comportement.

Dans l'ensemble, l'analyse de sensibilité montre que, bien que LEDBAT couvre une large gamme de niveaux de priorité (en particulier dans la troisième région T_3), son réglage est très peu pratique. En effet, les valeurs possibles en $T_3 \in [65, 100]\%$ sont très limitées, ce qui signifie qu'une petite variation de TARGET peut conduire à des scénarios complètement différents, où soit LEDBAT soit TCP Reno transmet le plus. En outre, les valeurs réelles de T dépendent des paramètres du réseau (par exemple, la capacité C , la taille du tampon B) et sont ainsi susceptibles d'être affectés par d'autres facteurs (par exemple, nombre de flux TCP Reno, hétérogénéité des RTT, etc).

Comparaison des protocoles LBE

Afin d'évaluer l'agressivité des protocoles LBE par rapport à TCP, nous considérons un scénario typique où N ($N \in [1, 10]$) flux de faible priorité (par exemple, en raison de services P2P ou autre) partagent le même goulot d'étranglement avec une seule connexion TCP, qui est représentatif d'un service générique de haute priorité, pour un total de $N + 1$ flux. À titre de référence, nous avons également simulé le cas où $N + 1$ flux TCP partagent le même goulot d'étranglement.

Pour comparer les protocoles LBE, nous utilisons les trois métriques présentées dans la section précédente, c'est à dire l'efficacité η , l'équité F et la probabilité de perte des paquets P_l .

Les résultats obtenus sont présentés dans Fig. 9. Compte tenu de l'efficacité η dans Fig. 9-(a), on voit que les deux protocoles basés sur le délai, NICE et LEDBAT, sont capables d'utiliser pleinement la bande passante laissée libre par TCP. Au contraire, dans le cas du protocole TCP-LP ou TCP Reno, les pertes de paquets entraînent une réduction d'efficacité. Si on regarde dans Fig. 9-(b) nous pouvons voir que l'équité F dans le cas de LEDBAT et NICE se rapproche de la valeur minimale possible (cet à dire, la région grise, qui indique les valeurs que l'équité ne peut pas atteindre, car ils sont plus petits que la limite inférieure $\frac{1}{N+1}$). Enfin, la probabilité de perte de paquets P_l est indiquée dans Fig. 9-(c). Encore une fois, les protocoles de contrôle de congestion fondés sur les délais et ceux fondés sur les pertes sont remarquablement différents : en dépit de son objectif de faible priorité, le montant de la perte induite par TCP-LP est étonnamment similaire à celui de TCP Reno classique. Par contre, la P_l dans le cas des protocoles NICE et LEDBAT est similaire et proche au minimum possible, où les pertes sont entamées par le flux TCP Reno présent

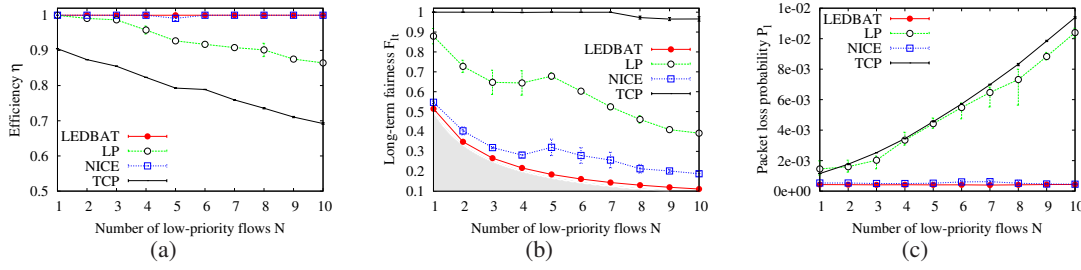


Figure 9: LBE contre un flux TCP : impact du nombre de flux LBE sur le performance de système.

sur le lien.

Après cette première évaluation nos conclusion sont que LEDBAT est un bon protocole, qui est capable de respecter ses objectifs : fournir un service à basse priorité, qui ajoute seulement un petit délai dans la file d'attente, mais qui est en même temps capable de profiter de la capacité disponible sur le lien. Notre étude comparative montre que LEDBAT réalise la priorité la plus faible possible par rapport à NICE et TCP-LP. En outre, nous constatons que le protocole TCP-LP hérite de sa conception (basée sur les pertes) une plus grande agressivité que NICE (basé sur le délai), dont le degré de priorité se trouve donc entre LEDBAT et TCP-LP.

Amélioration de LEDBAT pour l'équité

Nous avons vu dans la section précédente que LEDBAT est capable de céder les ressources à TCP et de partager équitablement le goulot d'étranglement dans le cas où plusieurs flux LEDBAT débutent en même temps.

Malheureusement, la situation change radicalement si nous faisons commencer les flux à deux instants différents : dans ce cas là, le comportement observé dépend fortement du délai entre les deux flux, comme nous pouvons observer dans les trois graphiques de Fig. 10-(a). Dans celui d'en haut, le deuxième flux commence quand le premier n'a pas encore commencé à occuper la file d'attente. Du coup, bien que les deux flux aient bien mesuré le délai de propagation, le premier flux ayant commencé d'abord, arrive à occuper une partie de la bande disponible bien plus importante que le deuxième. Dans le graphique du milieu, le deuxième flux commence quand le première a déjà atteint le TARGET : pour cela, le deuxième surestime le délai de propagation de TARGET ms et commence à augmenter sa fenêtre de congestion. Vu que le premier flux diminue son débit à la même vitesse à laquelle le deuxième augmente, celui-ci n'a pas la possibilité de corriger sa mauvaise estimation : finalement le premier flux se tais complètement alors que le deuxième continue à monter. Heureusement, à un certain moment le tampon est saturé et il y a une perte qui cause une brusque réduction de la fenêtre de congestion et le vidage de la queue. Après cette évènement, le deuxième flux corrige l'estimation du délai de propagation et l'équité est récupérée. Toutefois, si le tampon avait été assez grand pour pouvoir contenir une longue

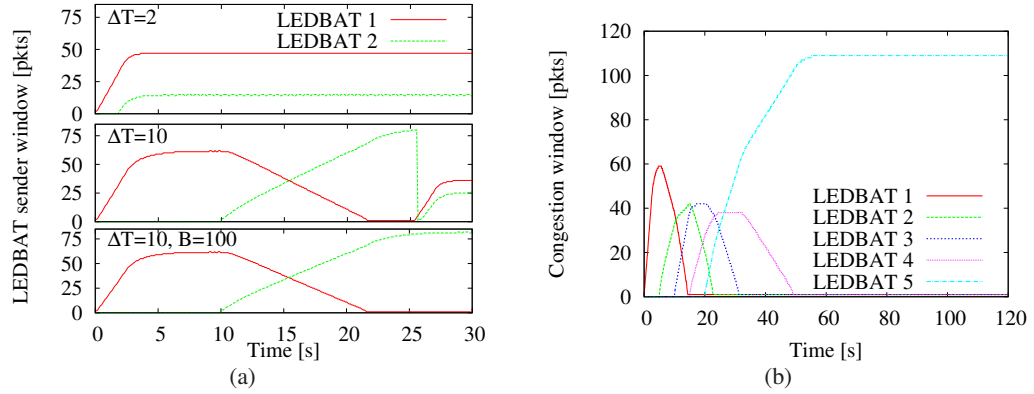


Figure 10: LEDBAT vs LEDBAT : Évolution temporelle de la fenêtre de congestion pour différentes conditions initiales et le phénomène du *latecomer advantage* avec deux ou plus flux.

queue, alors nous nous serions retrouvés dans la situation du graphique d'en bas, où nous voyons qu'une très mauvaise répartition de la bande passante peut persister longtemps. En Fig. 10-(b) nous voyons que ce phénomène peut se présenter aussi quand plusieurs flux insistent sur le même lien, à conditions que le tampon soit assez grand pour contenir la file d'attente créée.

Une objection à nos simulations, qui a été faite sur la liste de diffusions du groupe de travail IETF, était que ce phénomène ne peut pas se présenter dans un véritable réseau, parce que les retards aléatoires introduits par le système d'exploitation, les routeurs et d'autres trafics ne laisseraient pas les deux flux se synchroniser autant. Si, dans une certaine mesure, il est vrai qu'il y a beaucoup moins de synchronisation dans un réseau réel, ce phénomène de *latecomer advantage* peut quand même survenir. Afin de démontrer cela, nous avons profité de la publication du code officiel de LEDBAT [49], pour tester la même situation sur un testbed réel. Les résultats que nous avons mesurés sont présentés en Fig. 11 et ils sont tout à fait similaires à nos simulations. Nous voyons que le deuxième flux fait taire le premier car il surestime le délai de propagation, comme nous pouvons remarquer dans le plot à droite que l'offset calculé du TARGET est toujours égal à zéro pour le deuxième flux.

Par contre, il y a un problème dans son design originel qui porte à une situation de partage non équitable des ressources quand deux flux insistent sur le même lien. Nous allons étudier ce problème dans le reste de cette section, en proposant aussi des solutions efficaces.

Au cours de notre étude, nous avons découvert que la raison principale de l'iniquité de LEDBAT doit être recherchée dans la spécification du contrôleur de l'algorithme de contrôle de congestion, et, en particulier, dans la composante de décroissance additive. Déjà, dans les dernières années 80, Jain [32] avait remarqué qu'un algorithme de contrôle de congestion ayant cette composante était instable et incapable de converger vers un équilibre : il indiquait la décroissance multiplicative comme un élément essentiel à l'équité de l'algorithme.

Nous avons défini dans le Chap. 4 un nouveau protocole, fair-LEDBAT ou fLEDBAT, qui ne

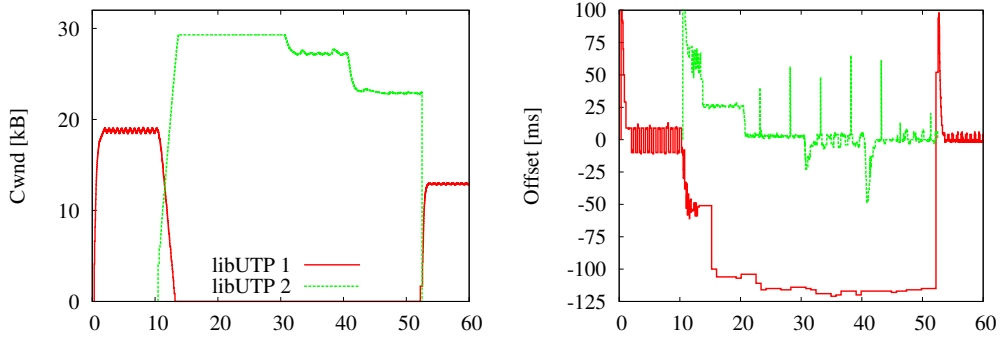


Figure 11: Testbed LAN expérimentale : évolution de la fenêtre de congestion (en haut) et offset par rapport au target (en bas) pour deux flux concurrents libUTP.

réintroduit non seulement la décroissance multiplicative dans LEDBAT pour atteindre l'équité, mais apporte aussi d'autres petites modifications afin d'améliorer l'efficacité du protocole. Nous commençons par reprendre la spécification de LEDBAT sous une forme plus formelle : si on dénote avec D_{min} le `base_delay`, avec τ le `TARGET`, avec $q(t)$ le délai dû à la file d'attente au temps t et avec $cwnd(t)$ la fenêtre de congestion au temps t , nous pouvons exprimer le contrôleur de LEDBAT comme :

$$\begin{aligned} \Delta(t) &= (q(t) - D_{min}) - \tau \\ cwnd(t+1) &= \begin{cases} cwnd(t) + \alpha \frac{\tau - \Delta(t)}{\tau} \frac{1}{cwnd(t)} & \text{sans pertes,} \\ \frac{1}{2} cwnd(t) & \text{en cas de perte.} \end{cases} \end{aligned}$$

Par contre fLEDBAT est spécifié comme ci après, où α et ζ sont les paramètres de l'algorithme.

$$cwnd(t+1) = \begin{cases} cwnd(t) + \alpha \frac{1}{cwnd(t)} & \text{sans pertes et } \Delta \leq 0, \\ cwnd(t) + \alpha \frac{1}{cwnd(t)} - \frac{\zeta}{\tau} \Delta & \text{sans pertes et } \Delta > 0, \\ \frac{1}{2} cwnd(t) & \text{en cas de perte.} \end{cases}$$

Nous voyons que nous avons apporté une double addition : (i) nous avons ajouté une terme additive avec α qui a l'objectif d'améliorer l'efficacité de l'algorithme et (ii) nous avons réintroduit un terme multiplicatif avec ζ quand le délai d'attente dépasse le target délai τ . Dans le chapitre Chap. 4 nous démontrons avec un modèle fluide du trafic que l'iniquité de LEDBAT est en effet dû à sa composante additive et, par la suite, que fLEDBAT a de bonnes propriétés non seulement d'équité mais aussi d'efficacité et de convergence. Nous avons implémenté le nouveau protocole dans le simulateur `ns2` et nous avons aussi simulé notre modèle fluide numériquement: dans la

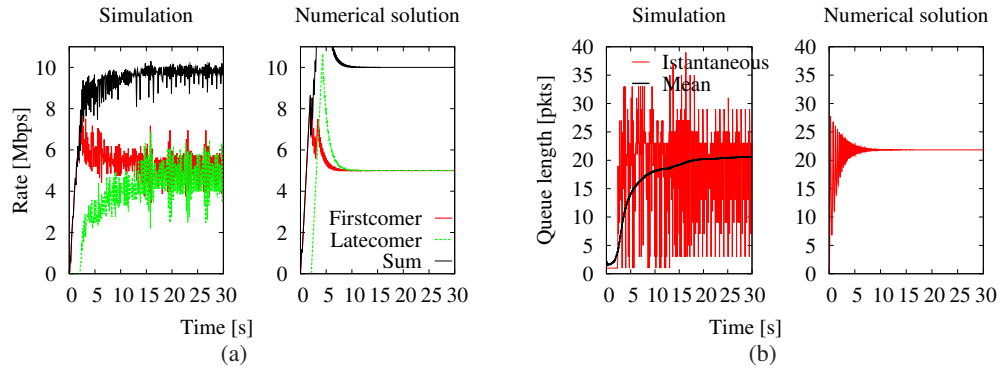


Figure 12: Comparaison de (gauche) simulation et (droite) solution numérique pour (a) débits et (b) dimension de la file d’attente.

Fig. 12 nous montrons les résultats de la simulation et de la solution numérique, où nous voyons que l’équité est rétablie et qu’il y a une parfaite correspondance entre la simulation et le modèle.

Dans le Chap. 4 nous conduisons une analyse très fine du nouveau protocole. Nous effectuons d’abord un tarage des paramètres α et ζ dans des scénarios avec un et plusieurs flux, pour comprendre l’intervalle des valeurs qui permet d’obtenir les meilleurs performances. Ensuite, nos simulations se concentrent sur le modèle de trafic (*backlogged* ou par *chunk*) et sur des scénarios mult flux et multipaires, pour évaluer le protocole dans des conditions d’utilisation typiques, vu que LEDBAT a été conçu pour être le protocole par défaut de BitTorrent. Dans tous les cas, fLEDBAT se comporte mieux que LEDBAT et atteint une meilleur performance ainsi qu’une équité supérieure.

Impact de LEDBAT sur BitTorrent

Étude de simulation

Jusqu’à présent, nous avons concentré notre attention sur l’étude du comportement LEDBAT dans des scénarios relativement simples (par exemple, un seul goulot d’étranglement, multiple flux, etc), qui sont nécessaires pour dévoiler les performances du point de vue débit, mais qui sont également assez éloignés des conditions P2P dans lequel le protocole est déployé. Par ailleurs, une analyse préalable de ce type donne généralement des réponses sur des questions telles que l’équité et l’efficacité, qui sont naturelles dans une perspective de contrôle de congestion, mais ne sont pas directement en rapport avec les performances perçue par les utilisateurs des systèmes P2P.

Dans Chap. 5, nous affinons la compréhension de LEDBAT, en mesurant son impact sur la métrique primaire pour les utilisateurs de BitTorrent, notamment le temps de téléchargement du contenu, au moyen de simulations au niveau paquets sur ns2. Une étude expérimentale sur les

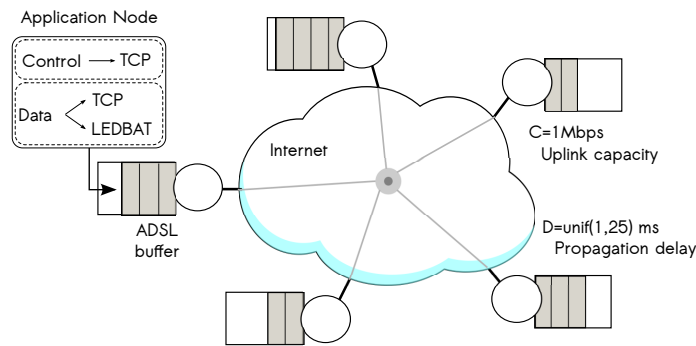


Figure 13: Synoptique du modèle de simulation.

mêmes questions sur des réseaux réels est présenté dans Chap. 6. Les résultats de nos enquêtes montrent que: (i) dans le cas où les clients LEDBAT remplacent totalement les clients TCP, aucune différence de performance n'est visible; (ii) en cas de population hétérogène, comprenant des peers qui utilisent LEDBAT et d'autres TCP, les peers qui utilisent LEDBAT obtiennent un temps de finalisation plus faible, grâce au délai d'attente inférieur sur le lien montant, qui bénéficie au trafic de signalisation (par exemple, les demandes de chunk).

Pour réaliser notre étude, nous avons recours à des implémentations open-source de BitTorrent [2, 41] et LEDBAT [7] pour ns2. Néanmoins, certaines modifications ont été nécessaires afin de réussir l'intégration de deux modules, ce que nous allons résumer brièvement par la suite, à l'aide de Fig. 13.

Nous mettons en œuvre deux types de peers P2P, selon le protocole de congestion utilisé sur l'*uplink*, donc pour l'envoi des données. Nous supposons que les dernières versions de l'application BitTorrent vont initier, par défaut, des transferts de données en utilisant LEDBAT sur le lien montant, mais qu'ils acceptent aussi des connexions TCP entrantes. De même, les anciennes versions vont initier des transferts sur TCP, mais peuvent aussi accepter des connexions LEDBAT entrantes. En outre, dans tous les cas, les applications utilisent le protocole TCP pour l'échange des messages de signalisation. Par conséquent, un mélange de trafic différent peut-être en concurrence sur le lien ADSL et dans la mémoire tampon.

Nous simulons un scénario (*mid*) *flash-crowd* dans lequel, au temps $t = 0$, le swarm torrent est constitué par un seul seed. Dans un deuxième temps, 100 peers vont rejoindre le seed avec des temps d'arrivée ayant une distribution exponentielle (taux moyen de 0.1 Hz). En ce qui concerne la population du swarm, nous considérons deux scénarios homogènes (soit uniquement peers sur TCP ou uniquement peers sur LEDBAT) et un scénario hétérogène, où la population est divisée également, en moyenne, entre peers LEDBAT et TCP (désigné comme 50-50). Enfin, en ce qui concerne le comportement des peers et des seeds, nous considérons trois scénarios différents, à savoir (i) *never leave* (les seeds ne quittent jamais), (ii) *random stay* (les seeds restent un temps aléatoire), et (iii) *immediate leave* (les seeds partent immédiatement).

Principaux résultats

Les résultats de simulation sont présentés dans Fig. 14. Les figures en haut se réfèrent à des scénarios où les seeds ne quittent jamais le système, les figures au milieu à des scénarios avec un temps aléatoire et celles d'en bas à des scénarios avec départ immédiat. À gauche sont reportées les évolutions dans le temps au cours de la simulation, triées par rang d'achèvement des peers (la région en gris foncé représente la période initiale de transition non prise en compte). Si les seeds restent pour toujours, les temps de téléchargement vont se réduire à un point où les téléchargements sont proches d'exploiter pleinement de la capacité en downlink. Cela n'est plus le cas lorsque les seeds ne restent que pour un temps fini ou s'ils partent des qu'ils finissent le téléchargement.

Les figures au milieu de Fig. 14 montrent la distribution cumulative des temps de téléchargement (CDF) pour les différentes populations. Si les seeds ne partent jamais, il n'y a aucune différence entre différents algorithmes de contrôle de congestion, car il n'y a pas de ressources critiques. À l'inverse, lorsque les seeds quittent le système et sont remplacés par de nouveaux peers (scénarios *random stay* ou *immediate leave*), les ressources deviennent rares, ce qui se traduit par des temps de téléchargement plus longs. Notez que le temps de téléchargement dans ces deux derniers scénarios sont affectés par le mécanisme de contrôle de congestion adoptées par les peers, dans le cas de swarm hétérogènes, alors que les temps sont presque les mêmes dans le cas de swarm homogènes.

La raison pour laquelle cela arrive peut être mieux comprise en regardant la distribution cumulative complémentaire (CCDF) de la file d'attente, montrée dans les figures à droite de Fig. 14 (avec un échantillonnage des files d'attente de liaison montante à 10 Hz). Notez bien que la file d'attente de liaison montante de peers LEDBAT est très similaire dans les trois cas, vu que LEDBAT essaie de ne pas dépasser un TARGET délai : l'écart par rapport au valeur target est dû au trafic TCP qui partage la même file d'attente, et le *latecomer advantage*, présenté dans la section précédente. Au contraire, les files d'attente dans le peers TCP peuvent atteindre des valeurs plus élevées, jusqu'à plus d'une seconde de délai compte tenu des paquets de taille maximale. Dans le cas 50-50, le délai obtenu est la somme des délais dûs à LEDBAT et à TCP sur le même lien en uplink, résultant en un système de files d'attente intermédiaires entre les scénarios homogènes.

Fait intéressant, les temps de téléchargement changent si on considère un swarm hétérogène, comme le montre la figure du milieu dans Fig. 14-(b). Dans les faits, nous constatons que les peers LEDBAT ont un temps de téléchargement plus courts dans le scénario 50-50 *random stay*, comme la répartition LEDBAT vs TCP du temps de téléchargement (CDF) dans Fig. 15-(a) l'atteste, alors qu'aucune différence ne se pose dans le cas 50-50 *never leave*. Cela est un résultat contre-intuitif, car on ne s'attend pas à ce que le temps de téléchargement soit lié au contrôle de congestion utilisé pour gérer les *envoi* de chunks. Nous soupçonnons que ce phénomène inattendu se produise en raison de (i) le découplage de la connexion de contrôle et de données, associé à (ii) la très grande taille des tampons ADSL : les grands tampons sont bénéfiques aux connexions de données *backlogged*, mais peuvent nuire à la signalisation de BitTorrent.

Afin de confirmer cette intuition, nous avons réalisé une série de simulations en faisant varier la taille du tampon des liens d'accès. Les statistiques des temps de téléchargement (moyenne, 1er

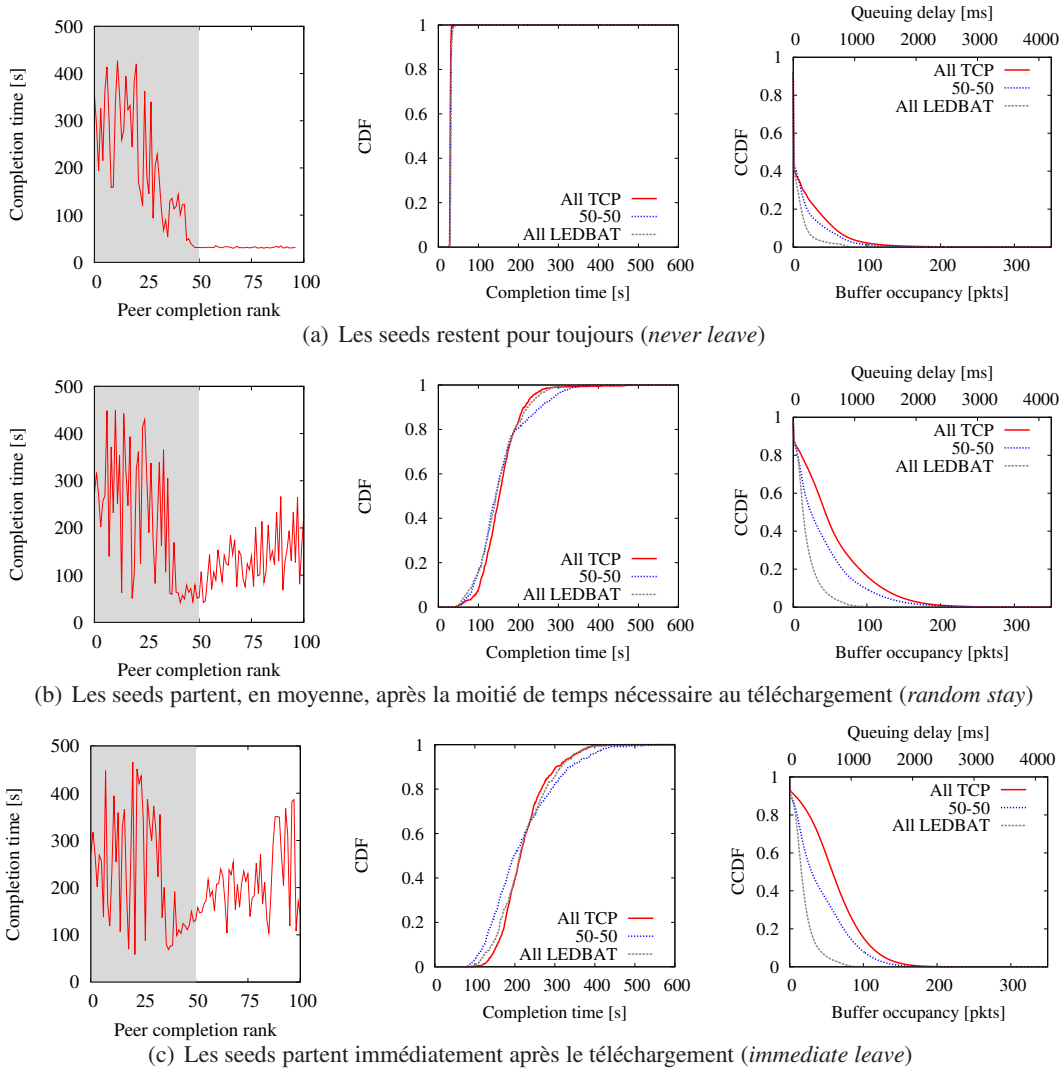


Figure 14: Performances des swarm pour différents temps de départs et populations (LEDBAT homogène vs TCP homogène vs hétérogène 50-50): évolution du temps de téléchargement et CDF, CCDF d'occupation du tampon.

et 3re quartile de la distribution) sont présentées sur Fig. 15-(b) pour le scénario 50-50 *random stay*, avec des courbes distinctes pour les peers LEDBAT et TCP. En accord avec les résultats présentés dans le chapitre Chap. 3 pour les performances au niveaux flux, LEDBAT est affecté par la taille du tampon seulement quand le délai target dépasse la taille du tampon. Dans ce cas, LEDBAT devient aussi agressif que TCP et son comportement est basé sur les pertes des paquets. Fig. 15-(b) confirme ce résultat. En particulier, si la taille de la mémoire tampon est trop petite par

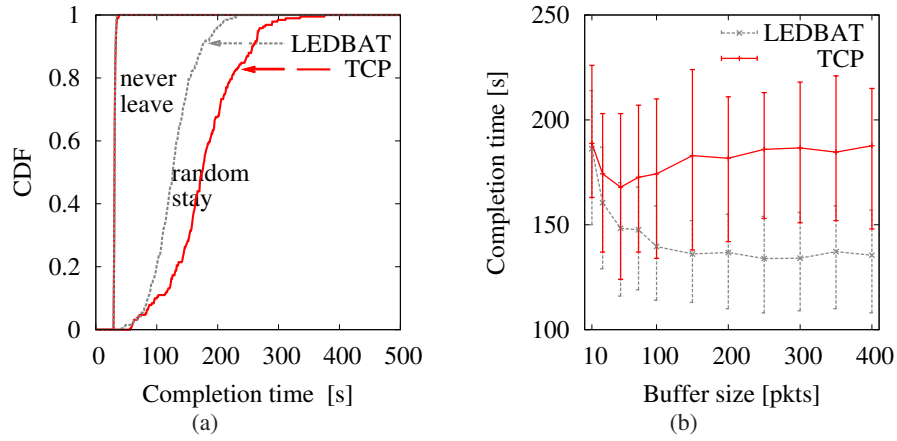


Figure 15: (a) Répartition du temps de téléchargement CDF en fonction de la population des peer dans le scénario hétérogène 50-50, pour différents comportement des seeds. (b) Temps de téléchargement du swarm (moyenne, 1er et 3er quartiles) en fonction de la taille du tampon B pour le scénario 50-50 *random stay*.

rapport au target, le rendement de LEDBAT et TCP ne peuvent pas être distingués. A l'inverse, LEDBAT et TCP montrent un comportement différent quand on augmente la taille du tampon : les transferts de données sur TCP bénéficieront de l'augmentation de la taille du tampon (au prix d'une signalisation plus lente), tandis que les transferts des données sur LEDBAT permettent de limiter le délai de transmission, quelle que soit la taille du tampon (et le temps de téléchargement est à peu près inchangée pour $B > 50$).

Étude de mesure

Notre étude précédente, présenté dans Chap. 5, suggère que LEDBAT est plus performant que TCP standard, vu que l'utilisation de LEDBAT limite pratiquement le délai d'attente à une valeur target, et cela se traduit par une accélération de signalisation comme un effet secondaire. Cependant, les résultats du chapitre précédent sont fondés uniquement sur des simulation *ns2* : il devient donc impératif d'évaluer si les phénomènes observés se passent aussi dans la pratique, ce qui est précisément l'objectif de l'étude réalisée dans ce chapitre.

L'impact de ce nouveau protocole sur les performances de BitTorrent peuvent être affectés essentiellement par deux paramètres différents. Au niveau du flux, LEDBAT est principalement affecté par la choix du délai target (*net.utp_target_delay*). Au niveau du swarm, la préférence des protocoles de transmission (*bt.transp_disposition*) de chaque peer entre LEDBAT et TCP joue un rôle important aussi. Par conséquent, avant d'exécuter un ensemble d'expériences, dans le chapitre Chap. 6, nous présentons quelques idées préliminaires sur ces deux paramètres dans le client utilisé.

Si le choix du paramètre *net.utp_target_delay* est fait de manière immédiate, le deuxième ne

mérite plus d'attention. En fait, *bt.transp_disposition* contrôle le protocole utilisé pour la connexion de données entrant et sortant du peer. Comme indiqué dans le manuel μ Torrent en ligne ¹, *bt.transp_disposition* est un masque à bit qui résume les comportements suivants :

- 1: ouvrir des connexions TCP sortantes
- 2: ouvrir des connexions LEDBAT sortantes
- 4: accepter les connexions TCP entrantes
- 8: accepter les connexions LEDBAT entrantes
- 16: utiliser le format nouvel en-tête LEDBAT

μ Torrent par défaut utilise la valeur 31, ce qui signifie que le client accepte les protocoles TCP et LEDBAT, soit pour envoyer ou recevoir des données, en utilisant éventuellement le format d'en-tête LEDBAT.

Toutefois, comme l'a aussi confirmé Arvid Norberg, l'un des principaux développeurs de BitTorrent, le client μ Torrent a une préférence *hardcoded* pour LEDBAT, de sorte que dans le cas où une connexion TCP et une connexion LEDBAT sont établies avec succès, la TCP sera fermée et seul la connexion LEDBAT sera utilisée. Comme nous le verrons dans le résultats, cette préférence a des conséquences importantes (et bénéfique) sur le temps de téléchargement global.

Population homogène

Afin d'estimer l'impact de ce paramètre, nous avons réalisé des expériences sur une plate-forme de machines dédiées qui exécutent Linux comme système d'exploitation et utilisent le client μ Torrent version 3.0. Les machines de la plate-forme Grid'5000 [5] sont inter-connectées par un réseau local à haut débit de 1 *Gbps*, sur lequel nous avons imposé des restrictions de bande passante et de délais d'attente ajoutées en utilisant le module *netem* pour Linux. Comme indiqué dans [79] et confirmé expérimentalement dans le chapitre Chap. 2 et par [78], les modems ADSL peuvent accommoder en mémoire tampon jusqu'à quelques secondes de trafic. Pour cette raison, dans nos expériences, nous avons mis la taille du tampon B en fonction de la capacité C du lien montant de sorte que $B/C = 1$ seconde de trafic. Plus précisément, nous présentons dans la suite des résultats pour $C = 1$ *Mbps* seulement.

Nous avons utilisé 76 machines afin de simuler un scénario Internet *flash crowd*, où un seul seed fournit, en premier lieu, tout le contenu (un fichier de 100 *MB*) pour un certain nombre de peers qui arrivent tous en même temps (et qui ne quittent jamais le système). Afin d'éviter les effets indésirables dus à l'influence mutuelle des peers multiples exécutées sur le même hôte, nous sommes obligés de mettre un seul peer sur chaque hôte de la plate-forme Grid'5000, donc pour un total de 76 peers.

¹http://www.bittorrent.com/help/manual/appendixa0212#bt.transp_disposition

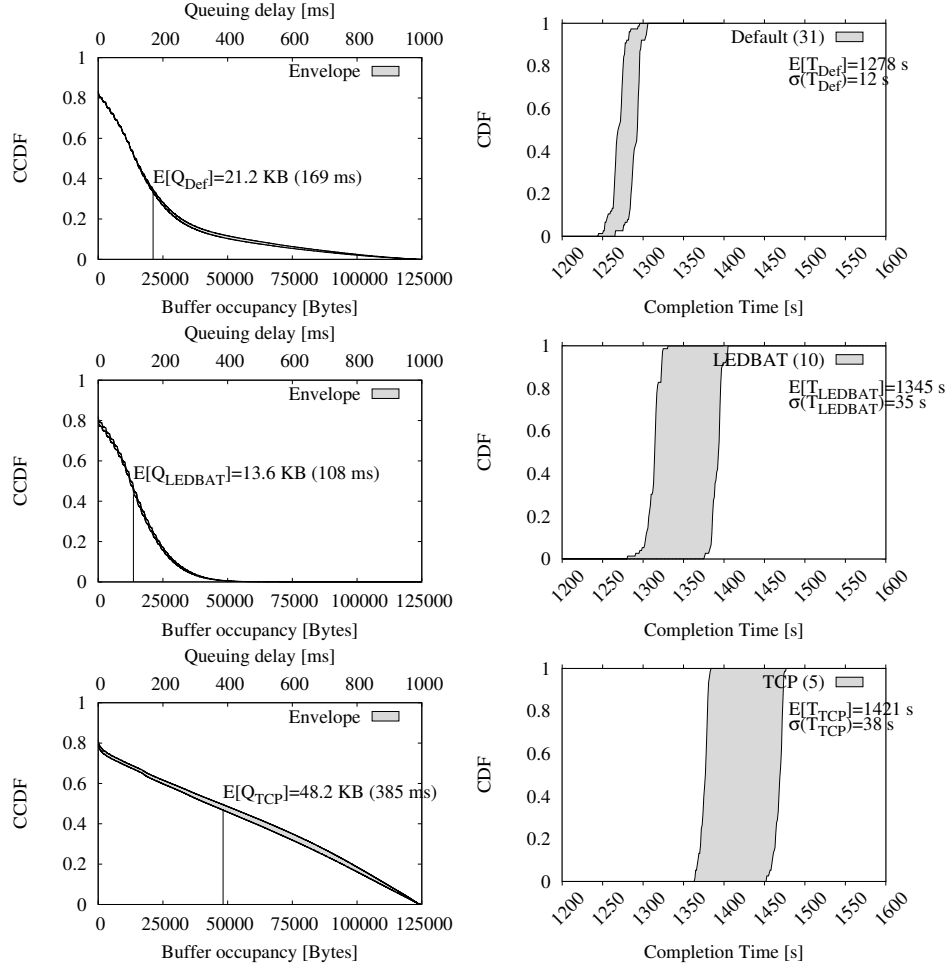


Figure 16: CCDF de l'occupation du tampon (à gauche) et CDF du temps de téléchargement (à droite) pour les swarm homogènes : réglages par défaut (*bt.transp_disposition*=31, en haut), swarm LEDBAT seulement (*bt.transp_disposition*=10, au centre) et swarm TCP seulement (*bt.transp_disposition*=5, en bas). La ligne verticale dans l'occupation du tampon représentent la moyenne de la longueur de la file d'attente (en KB et ms).

Les résultats dans le cas homogène sont présentés dans Fig. 16. Pour chaque métrique d'intérêt, la figure indique l'enveloppe des résultats recueillis, soit les courbes minima et maxima pour trois itérations. Nous exprimons les résultats en termes de (i) la fonction de distribution cumulative (CDF) du temps de téléchargement T à droite et de (ii) la fonction de distribution cumulative complémentaire (CCDF) de l'occupation du tampon Q du lien d'accès pour chaque peer à gauche. L'occupation du tampon est exprimée à la fois en octets (axe X inférieur) et en termes de la quantité

de délai qu'une application interactive subirait (axe X supérieur).

Dans la partie supérieure nous présentons un scénario dans lequel tous les peers utilisent les paramètres par défaut (*bt.transp_disposition=31*), c'est à dire, les peers sont capables de parler les deux protocoles TCP et LEDBAT. Au milieu nous rapportons le cas d'un swarm tout LEDBAT (*bt.transp_disposition=10*), alors que le swarm avec uniquement TCP (*bt.transp_disposition=5*) est représenté en bas.

Notez que, sur le long terme, tous les swarms atteignent une efficacité similaire : en regardant la CDF de l'occupation du tampon dans Fig. 16 nous pouvons voir que dans environ 80% du temps, après une opération de de-stockage, la file d'attente n'est pas vide. Cette efficacité est aussi liée à la dynamique du système BitTorrent (par exemple, pipeline des requêtes, dynamiques des échanges de chunks, etc.). Le nombre de paquets restants dans la file d'attente après une transmission de paquets dépend en outre du protocole de contrôle de congestion choisi. Comme prévu, la dynamique AIMD de TCP ont tendance à remplir la mémoire tampon, tandis que LEDBAT s'efforce (et réussit) pour limiter la taille de la file.

Notez que la taille de la file ne peut pas expliquer à elle seule la différence dans les statistiques de temps de téléchargement (car, sinon, le temps avec LEDBAT sera le plus faible). Par conséquence, nous conjecturons que ce résultat est la combinaison de deux effets sur le plan de contrôle et de données, qui sont assistés par l'utilisation de LEDBAT et TCP respectivement. Tout d'abord, une occupation majeure de la file d'attente en raison de TCP peut influencer négativement le temps de téléchargement, en faisant obstacle à une diffusion en temps et en heure des informations de contrôle (par exemple, le message *INTEREST*). Plus le temps nécessaire pour signaler les intérêts est grand, plus le temps avant le début du téléchargement et leur envoi à d'autres peer est long (ce qui nuit à tous le peer TCP).

Notez bien que, comme dans le cas homogène avec TCP, le délai peut atteindre jusqu'à 400 ms en moyenne, cela implique que le RTT pour la signalisation des échanges peut atteindre l'ordre d'une seconde, ce qui peut éventuellement ralentir considérablement la propagation des chunks.

Cependant, comme dit précédemment, les temps de téléchargement ne sont pas entièrement expliqués en termes de délai d'attente, sinon le swarm avec uniquement LEDBAT devrait être le gagnant. Pourtant, alors que LEDBAT limite la taille de la file et évite d'interférer avec une diffusion rapide de messages de contrôle, il est également, par conception, moins agressif que TCP. Par conséquence TCP peut être plus efficace pour partager rapidement *les chunks dans le plan de données*. Cela peut à son tour nuire au téléchargement effectué avec LEDBAT, qui est légèrement supérieur par rapport au parametres par default *bt.transp_disposition=31*.

Population hétérogène

Après avoir vu qu'un mélange de protocoles TCP et LEDBAT peut être bénéfique pour le temps d'exécution, dans le Chap. 6, nous avons étudié des swarms avec des peers soit TCP (*bt.transp_disposition=13*), soit LEDBAT (*bt.transp_disposition=14*). Dans ces cas, les peers préfèrent l'un des deux protocoles pour ouvrir une connexion, mais ils peuvent accepter n'importe quel autre connexions entrantes. Nous considérons trois scenarios, nommé 25/75, 50/50, et 75/25 (avec une notation X/Y,

où $X\%$ représente le pourcentage de peers préférant TCP sur leur liaison montante).

Les résultats montrent que, même si une petite quantité de trafic TCP est bénéfique pour réduire le temps de téléchargement du swarm, une grande quantité de TCP peut au contraire ralentir le téléchargement des torrents pour l'ensemble du système. Cependant, les temps de téléchargement de chaque scénario sont pratiquement les mêmes pour les peers LEDBAT et TCP (avec un léger avantage pour ce dernier). Ainsi, à la différence de notre travail de simulation présenté dans Chap. 5, nous n'observons pas une inégalité de temps de téléchargement entre les différentes classes de peers au sein d'un swarm hétérogène. Cela est dû au fait que l'analyse de la simulation précédente considérait un modèle plus simple pour *bt.transp_disposition*, qui ne prenait pas en compte ni (i) la possibilité, pour les peers TCP, d'utiliser une connexion LEDBAT déjà ouverte à l'envers, ni (ii) la préférence pour LEDBAT dans le code.

Conclusion

Dans cette thèse, nous avons étudié l'impact des protocoles de contrôle de congestion sur les applications pour le partage distribué de ressources. Plus précisément, nous concentrons notre attention sur le protocole LEDBAT, proposé par les développeurs de BitTorrent pour transporter le trafic de données dans leur service de distribution de contenu P2P. L'objectif principal de ce protocole est de mettre en place un transport *Lower-than Best Effort*, qui vise à réaliser un service à faible priorité qui (i) exploite la bande passante de réserve, c'est à dire, la bande passante en excès par rapport à la vitesse atteint par les flux TCP standard, (ii) sans interférer avec le trafic principal, et en particulier les flux interactifs.

Performance au niveau flux

Dans la première partie de la thèse, nous nous concentrons sur le point de vue de flux, afin d'évaluer la performance au niveau du transport de LEDBAT, un algorithme pour le contrôle de congestion à faible priorité basé sur le délai de transmission.

Dans ce but nous utilisons tout d'abord une approche expérimentale basée sur une méthodologie black-box afin d'étudier les performances du client μ Torrent. Nous exploitons un testbed local pour étudier le comportement de différentes versions du clients dans un ensemble de conditions réseau émuloées artificiels, comme des limitations de bande passante et des délais forcées sur le lien. Les scénarios abordés considèrent à la fois un flux unique, ainsi que de multiples flux en compétition entre eux sur le même goulot d'étranglement. Nos résultats montrent que LEDBAT est un protocole prometteur, capable de remplir ses objectifs de conception. Plus précisément, il utilise efficacement la bande passante laissée libre et limite l'effet *bufferbloat*, n'ajoutant que peu à la queue du tampon existant. D'après les tests effectués sur des liens ADSL et dans le scénario avec de multiples flux, nous découvrons que le protocole cède effectivement le passage à TCP. Toutefois, en fonction des paramètres de l'algorithme et des paramètres des trafics TCP en compétition, l'agressivité de LEDBAT présente un comportement différent.

Afin de poursuivre notre enquête, nous avons recours à la mise en œuvre du protocole LEDBAT

dans le simulateur du réseau `ns2`, en utilisant le Draft délivré par le groupe de travail IETF. Le protocole est décrit comme un algorithme basé sur le délai, qui utilise donc les modifications dans le délai unidirectionnel émetteur-récepteur comme des indications de congestion naissante sur le chemin. La fenêtre de congestion de l'émetteur est réglée au moyen d'un contrôleur linéaire, de manière à atteindre une quantité fixe de délai additionnel (qui correspond au paramètre `TARGET`) dans la files d'attente.

Les résultats des simulations confirment que cette conception conduit à un transfert de faible priorité, qui cède le passage aux autres flux TCP simultanés. Ensuite, nous comparons les performances de LEDBAT avec d'autres protocoles similaires dans leur approche, tels que TCP-LP et NICE. Nos résultats montrent que le protocole TCP-LP est le plus agressif des trois à cause de sa conception basée sur les pertes de paquet, tandis que LEDBAT obtient la priorité la plus basse parmi tous. Une analyse de sensibilité sur les paramètres de délai `TARGET` et `GAIN` a mis en évidence la difficulté de réglage du niveau d'agressivité de LEDBAT par rapport à TCP. Dans l'ensemble, si la variation de `GAIN` n'a presque aucune influence, l'impact de `TARGET` est plutôt difficile à contrôler, étant donné que même de petits changements dans cette valeur entraînent des variations abruptes de performance.

L'étude de deux, ou plus, flux LEDBAT qui commencent à des moments différents et concourent pour le même goulot d'étranglement, a révélé un problème d'équité, qui se révèle aussi bien dans les simulations, que dans les expériences sur des testbed ou sur des réseaux réel. Cette situation malheureuse permet le flux retardataire d'exploiter toute la capacité disponible, au détriment des transferts existants, qui sont obligés à interrompre le transfert. Comme LEDBAT peut être utilisé pour le trafic d'arrière plan, la normalisation d'un protocole défectueux peut avoir des effets potentiellement dramatiques, car cela peut sérieusement affecter les performances de longs transferts, comme par exemple pour les services de sauvegarde virtuelle à distance. Notre enquête a permis d'identifier la composante additive de LEDBAT comme la cause première de ce problème, comme l'a déjà souligné les travaux sur l'adaptation de la fenêtre de congestion [32]. Pour cette raison, nous présentons `fLEDBAT` ("fair-LEDBAT"), une modification du protocole qui résout le principal inconvénient théorique de LEDBAT tout en préservant ses objectifs initiaux. Nous avons prouvé, au moyen de techniques analytiques, que notre solution converge vers un point de travail équitable et efficace et avons mis en avant, grâce à un vaste ensemble de simulations, qu'elle fonctionne bien sous un large éventail de paramètres.

Impact de LEDBAT sur BitTorrent

Dans la deuxième partie de cette thèse, nous concentrons notre attention sur l'impact du protocole LEDBAT sur la performance de BitTorrent, en particulier sur le temps de transfert, la métrique qui définit le mieux la qualité de l'expérience ressentie par les utilisateurs.

D'abord nous avons recours à la simulation, à l'aide de notre implémentation de LEDBAT pour `ns2` mise en œuvre avec celle de BitTorrent déjà existante [2, 41]. Afin de saisir le comportement des peers avec différents degrés de diffusion du nouveau protocole, nous avons considéré plusieurs types de population, à la fois homogène (tous les peers utilisent le même protocole de transport) et hétérogènes (peers mixtes utilisant le protocole TCP et LEDBAT). En outre, nous considérons trois

stratégies différentes de *seeding*, avec des peers qui partent immédiatement après le téléchargement, partent après un temps aléatoire, ou restent pour toujours dans le système. Les résultats des simulations montrent que, dans un scénario de population hétérogène, lorsque les pairs restent pour un temps aléatoire ou lorsqu'ils quittent immédiatement le système, l'utilisation de LEDBAT mène à un temps plus petit de téléchargement. Cela est dû au fait que LEDBAT limite le délai d'attente à l'accès, ce qui permet à son tour de réaliser une signalisation plus rapide et éviter de s'auto-congestionner comme TCP. Pour évaluer l'impact du paramètre `TARGET` du point de vue de BitTorrent, nous avons monté un scénario dans lequel les peers ont des valeurs de target hétérogènes. Nos résultats montrent que les peers avec une valeur inférieure réalisent un temps de téléchargement plus faible.

Dans le même esprit, nous avons ensuite évalué l'impact de LEDBAT sur BitTorrent en utilisant une approche expérimentale sur le plate-forme Grid'5000 [5]. Nous avons utilisé le client μ Torrent pour Linux afin de régler le paramètre `TARGET` comme souhaité et pour aussi choisir le protocole de transport pour la transmission de données. Comme dans l'étude réalisée par moyen de simulations, les scénarios considèrent à la fois une population de peers homogènes où hétérogènes. Les résultats ont confirmé les constatations recueillies plus tôt, à savoir que LEDBAT peut aider à réduire le temps de téléchargement torrent. En outre, nous découvrons que des performances encore meilleures peuvent être réalisés avec un mélange précis de trafic TCP et LEDBAT, comme dans le cas de la configuration par défaut du client BitTorrent. Dans ce cas, le peer qui bénéficie de la signalisation plus rapide grâce à LEDBAT et de la grande efficacité dans le plan de téléchargement grâce à TCP, peut atteindre les meilleures performances.

Dans l'ensemble, les principales contributions de cette thèse constituent une étape vers la compréhension de LEDBAT comme protocole à faible priorité et du contrôle de congestion appliqué aux applications de distribution de contenu P2P en général. Tout d'abord, nous avons identifié un problème critique d'iniquité intra-protocole de l'algorithme LEDBAT original et nous avons proposé une solution modifiée, qui est capable de résoudre le problème sans affecter la réalisation de ses objectifs initiaux. En outre, nous avons évalué l'impact du protocole LEDBAT sur les performances BitTorrent, découvrant qu'il peut être bénéfique pour l'amélioration de la qualité de l'expérience des utilisateurs finaux. Cela soit en termes d'augmentation de l'interactivité (grâce au l'ajout limité de délai), soit en termes de téléchargement de contenu de plus courte durée.

Contents

1	Introduction	5
1.1	LEDBAT: specification and evolution	8
1.2	The BitTorrent protocol in a nutshell	11
1.3	Related work	14
1.3.1	Congestion control	14
1.3.2	Peer-to-Peer applications	16
1.3.3	Recent LEDBAT work	18
1.4	Contributions of this thesis	19
I	Congestion control perspective	23
2	Performance evaluation - Black box experiments	25
2.1	Methodology and Preliminary Insights	26
2.2	Single flow scenario	28
2.3	Multiple flows scenario	30
2.4	Summary	32
3	Performance evaluation - Simulation based	35
3.1	LEDBAT overview	36
3.1.1	Queuing Delay Estimate	36
3.1.2	Controller Dynamics	37
3.1.3	TCP Friendliness Consideration	38
3.1.4	Lower-than Best Effort transport protocols	39
3.2	LEDBAT performance	41
3.2.1	Reference scenario	41
3.2.2	Evaluation metrics	42
3.2.3	Homogeneous Initial Conditions	43
3.2.4	LEDBAT Sensitivity Analysis	45
3.2.5	LBE protocols comparison	48
3.2.6	LBE against LBE	49

3.2.7	Impact of RTT heterogeneity	51
3.3	Summary	51
4	Rethinking LEDBAT for fairness	53
4.1	Related work	53
4.2	Motivation	54
4.2.1	Latecomer in simulated environment	54
4.2.2	Latecomer in real networks	56
4.2.3	Current LEDBAT fairness issues	57
4.2.4	Impact of additive decrease	57
4.3	Proposed LEDBAT modification	59
4.3.1	Fluid model description	59
4.3.2	Fluid system dynamics	61
4.3.3	Main results	61
4.4	Simulation overview	64
4.5	Impact of traffic model	66
4.5.1	Chunk-by-chunk transfer	66
4.5.2	Backlogged transfer	67
4.6	Sensitivity analysis	68
4.6.1	fLEDBAT vs TCP	68
4.6.2	fLEDBAT vs LEDBAT	69
4.6.3	fLEDBAT vs fLEDBAT	69
4.6.4	Tuning ζ for multiple-flows scenario.	69
4.7	Appetizer to P2P scenarios	70
4.7.1	Single peer perspective	71
4.7.2	Entire swarm perspective	73
4.8	Summary	74
II	BitTorrent swarm perspective	77
5	Impact of LEDBAT on BitTorrent - Simulative approach	79
5.1	Methodology and Scenario	80
5.2	Main results	81
5.2.1	Swarm population and seed behavior	82
5.2.2	Target heterogeneity	85
5.3	Summary	86
6	Impact of LEDBAT on BitTorrent - Experimental approach	89
6.1	Preliminary Insights	89
6.1.1	<i>net.utp_target_delay</i> : Target delay settings	91

6.1.2	<i>bt.transp_disposition</i> : TCP vs uTP settings	91
6.2	Experimental Results	92
6.2.1	Grid'5000 Setup	93
6.2.2	Homogeneous <i>bt.transp_disposition</i> settings	95
6.2.3	Heterogeneous <i>bt.transp_disposition</i> settings	97
6.2.4	LEDBAT vs TCP in a nutshell	99
6.3	Summary	99
7	Conclusion	101
7.1	Summary	101
7.2	Future work	103
	Appendices	106
A	List of publications	107
	Bibliography	115

Chapter 1

Introduction

As recently pointed out in [30], “Internet delays now are as common as they are maddening”. With the increased use of bandwidth-intensive services, such as video streaming, remote data backup and distributed file-sharing, performance of interactive applications has encountered an un concealed deterioration. As a consequence, end users sitting at the edge of the network suffer from high and variable delays, that hamper the functioning of latency sensitive application (i.e., VoIP, Web browsing, network gaming, remote administration) and may trigger timeouts of fundamental protocols as DNS.

The root cause for these delays can be identified with the excess buffering inside a network, which is nicknamed “bufferbloat”. Although this issue is well-known, since its first study in [31], recent years have witnessed a widely spread buffer overprovisioning, result of reduced memory costs, that exacerbated the problem. This increase in buffering capability in network devices has the benefit of reducing packet losses, but on the other hand defeats the TCP loss-based design, that relies on *timely* packet drops to detect congestion in the path. With excessive buffering, packets pile up in big queues whenever they encounter a fast-to-slow transition in the network, that generally happens on the uplink from the subscribers to the core, where the fast local area connections hits the ADSL or Cable modems for Internet access. Those queues inevitably need time to drain and this translates into slowed response and significant delay, that can grow up to few seconds [16, 58].

To address this issue, there have been various propositions, that essentially cluster in two orthogonal directions. On the one hand, some researchers [30] recommend *local active queue management (AQM)* techniques (i.e., selective packet dropping or marking) as the ultimate solution to reduce queuing delay. On this subject, numerous algorithms were proposed over the year, from SFQ [65] and RED [44] in early 90’s, to CoDel [70], the last arrived on the scene. However, these proposals faced so far a rather limited adoption [40], mainly due to flawed design [24, 73] or the difficulty in tuning the parameters (as in RED [33]). On the other hand, we find works heading towards the engineering of *end-to-end flow and congestion control (CC)* protocols alternative to TCP. Those protocols may have different goals, such as controlling the streaming rate over TCP connections as done by YouTube or Netflix, or aggressively protecting user *Quality of Experience (QoE)*

as done by Skype over UDP, or to provide bulk transfers service such as Picasa background upload option, Dropbox synchronization or perform operative system updates, as in Microsoft Windows Update. In particular, for this latter branch of low-priority, background services, one finds several proposals in literature that target a *Lower-than Best Effort (LBE)* behavior, such as TCP-LP [60], NICE [81], 4CP [63], Microsoft BITS [12], and more recently, LEDBAT [79].

LEDBAT, named after *Low Extra Delay Background Transport*, is a congestion control protocol for the data transmission, working on top of UDP. It has been introduced in late 2008 by BitTorrent [69], the popular P2P file sharing system with hundreds of millions of daily users worldwide. However, the idea of a P2P protocol over UDP has been initially reckoned a reason for dramatic Internet meltdown and panic spread across popular websites [21]. Yet, as already discussed at IETF [8] and later recognized on the Web [13], the BitTorrent development process embraces both ISP-friendliness (through AS-aware peer selection process) and TCP-friendliness (through a novel congestion control protocol for data transfer).

The novel insight in the widely explored congestion control landscape is, in this case, the reasonable assumption that the bottleneck is most likely at the access of the network (e.g., at the ADSL modem line), which means that congestion is therefore typically *self-induced* by concurrent traffic sessions generated by the same user (e.g., BitTorrent transfers in parallel with Skype call and Web browsing). As a consequence, due to the large buffer adopted by access nodes, congestion provokes an increase in buffering delay, thus worsening the bufferbloat phenomenon.

This new protocol is designed to avoid introducing excessive delay due to buffer bloating and targets (i) efficient but (ii) low priority transfers. When LEDBAT flows have the exclusive use of the bottleneck resources, they fully exploit the available capacity. When instead other transfers –such as VoIP, gaming, Web or other TCP flows– are ongoing, LEDBAT flows back off to avoid harming the performance of interactive traffic. To attain the efficiency aim, LEDBAT flows need to create queuing, as otherwise the capacity would not be fully utilized. At the same time, due to the LBE aim, the amount of extra queuing delay caused by LEDBAT flows should be small enough to avoid hurting the interactive traffic. This goal is achieved by reacting earlier than TCP to congestion notification, and reducing its transmission rate so to avoid harming current traffic: while TCP infers congestion from packet losses, LEDBAT infers congestion from increasing buffering delay, hence prior than losses occur. Thus, another important contribution of LEDBAT is that it constitutes a relief for operators too, as they no longer need to throttle the now gentle P2P traffic [4].

Before looking into the details of the protocols operations and our experiment, let us linger over the actual diffusion of the LEDBAT protocol in the Internet. Despite recent research showing an increasing importance of video over the share of Internet traffic, BitTorrent still represents a significant portion of user generated data. According to the Cisco forecast, the P2P file transfer accounted in 2011 for up to 4,6 petabytes and will be more than double within 4 years [34]. Moreover, due to the recent shutdown of popular file-hosting services such as Megaupload/Megavideo [45], we can expect the Internet ratio of file-sharing to increase again.

In Fig. 1.1, we depict the BitTorrent traffic share (UDP and TCP traffic, overall traffic) over the last few year time-window. The data traffic is averaged at the PoP of 5 European ISPs that are continuously monitored within the NAPA-WINE project [42, 47]. The green dashed line represents

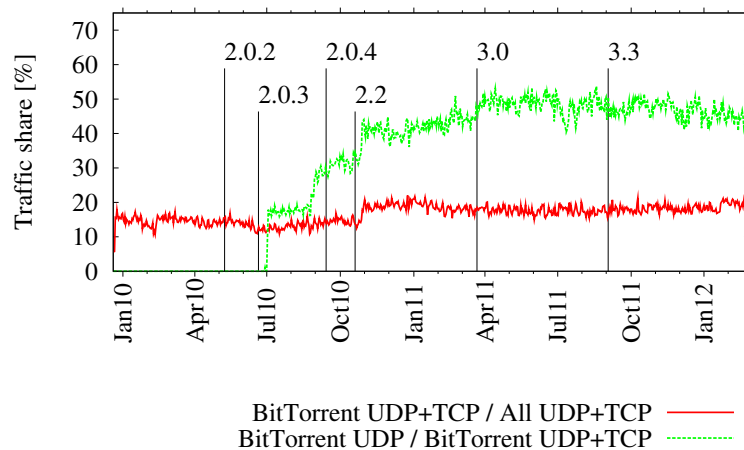


Figure 1.1: Proportion of BitTorrent traffic and of BitTorrent LEDBAT in the wild.

the relative percentage of BitTorrent traffic carried over UDP (hence, over the LEDBAT transport protocol), normalized over the total amount of BitTorrent traffic. Labels report a few BitTorrent application releases over the considered period¹. The figure clearly shows, soon after the release of μ Torrent 2.0.2, the first stable version to use data transport over LEDBAT, a steep increase of the percentage of BitTorrent traffic carried over UDP. Notice also that, while the overall share of BitTorrent is steady during the whole period², the percentage of data sent over UDP slowly increased and stabilized to about half of the BitTorrent traffic volume. This trend is confirmed also by the developers of BitTorrent, as Bram Cohen himself said, LEDBAT is “now the bulk of all BitTorrent traffic, [...] most consumer ISPs have seen the majority of their upload traffic switch to a UDP-based protocol” [36]. Such a great diffusion of the protocol highlights the importance of this work, that aims to study the inner mechanisms of LEDBAT, verify the validity of its goals and shed light on the advantages and weaknesses of the design choices.

The rest of this section is organized as follows. Sec. 1.1 provides a description of the LEDBAT protocol as it is delineated in the Draft specifications, while Sec. 1.2 presents an overview on the main aspects of the BitTorrent protocol. Sec. 1.3 places this work in the context of the related effort, and lastly Sec. 1.4 summarize the contributions of this thesis.

¹See “Announcement” thread from the μ Torrent forum <http://forum.utorrent.com/viewforum.php?id=4>

²With an increase after Megaupload shutdown, though the raise of P2P traffic was already anticipated in [42]

1.1 LEDBAT: specification and evolution

A few months ago, a post in the thread announcing the new μ Torrent release 1.9-alpha-13485 in the BitTorrent developer forum [69] raised a lot of motivated interest as well as quite a few unmotivated buzz [13]. Not only would the official BitTorrent client no longer be open-source, but it was, above all, introducing a novel “micro transport protocol” (uTP), a new application-layer protocol for data transfer implementing a innovative congestion-control algorithm and exploiting UDP at the transport layer.

Nevertheless, the main item retained was that BitTorrent would have switched its data transfer over UDP – which does not implement any kind of congestion control and is thus usually associated with *unresponsive* source. This fallacious interpretation raised serious concerns: as BitTorrent constitutes a significant portion of nowadays Internet traffic, its switchover to UDP was seen as the cause for the forthcoming collapse of the network. This “Internet meltdown” buzz rapidly flooded the Internet, and eventually slowed down only after an official reaction of BitTorrent [13], followed by intense discussions.

Yet, the buzz was not built on solid technical foundation: in fact, the original announcement [69] clearly stated that the design goal of the new protocol was to avoid “*killing your net connection – even if you do not set any rate limits,*” and to be able instead to “*detect problems very quickly and throttle back accordingly so that BitTorrent doesn’t slow down the Internet connection and Gamers and VoIP users don’t notice any problems.*” The inner working of this novel protocol is under discussion as BitTorrent Enhancement Proposals BEP-29 [71] as uTP, as well as IETF Low Extra Delay Background Transport (LEDBAT), whose first draft [79] has been accepted as a WG item in August 2009. Since the first release of the Draft, several modifications were made, however slight discrepancies exist between this document and the BEP-29, mainly concerning the parameter settings. In the reminder of the thesis we will adhere to the LEDBAT flavor available at the time of the analysis, as well as to the IETF appellation.

Fig. 1.3-(b) reports a timeline with the release dates of the Draft specification, from the original version to the last one, recently sent to the IESG (Internet Engineering Steering Group) for approval as a IETF RFC. The most relevant versions of the μ Torrent client used in the reminder of this thesis

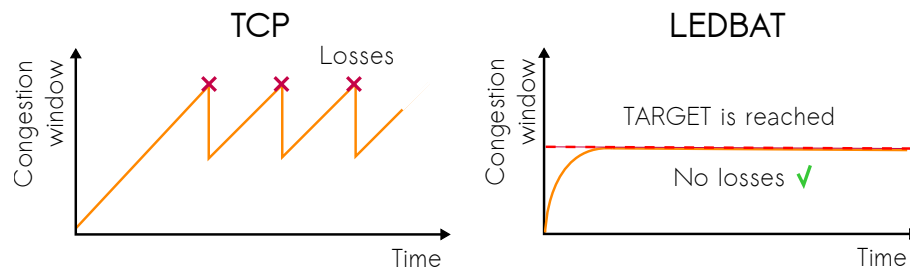


Figure 1.2: Comparison of the congestion window evolution in time of TCP and LEDBAT.

Table 1.1: Compendious of LEDBAT implementations.

Name	Language	Works on	Open-source	Reference
LEDBAT	C++/C	ns2 and Linux Kernel	Yes	[7]*
libUTP	C++	multi-platform	Yes	[49]
libtorrent	C++	multi-platform	BSD-licence	[9]
LEDBAT	C	Mac OS X	APSL v2.0	[1]
LEDBAT	Python	multi-platform	No	[19]
μ Torrent	C++	multi-platform	No	[69]
DW-LEDBAT	C++	ns2	No‡	[14, 15]
LEDBAT	unknown	unknown	No	[78]

* Our own implementation.

‡ A list of the modifications to existing files of ns2 is available in [14].

are reported in the bottom plot Fig. 1.3-(c).

LEDBAT is described in [79] as a windowed protocol, governed by a linear controller designed to infer earlier than TCP the occurrence of congestion on a network path. Its congestion control algorithm is based on the One-Way Delay (OWD) estimation: queuing delay is estimated as the difference between the instantaneous delay and a base delay, taken as the minimum delay over the previous observations. Whenever the sender detects a growing OWD, it infers that queue is building up and reacts by decreasing its sending rate. This way, LEDBAT reacts earlier than TCP, which instead has to wait for a packet loss event to detect congestion. At the same time, LEDBAT tries to add only a fixed amount of queuing delay (TARGET parameter of the algorithm) into the buffer, as depicted in the visual comparison in Fig. 1.2. A more accurate description of the LEDBAT algorithm, which include the pseudocode of the main functions and further details about its principal parameters and the controller dynamics, is reported in Chap. 3.

Since the first release of the Draft specification, several implementations of the LEDBAT protocol have been made, that we summarize in Tab. 1.1 with the details about the programming language used, the platform for which are built and the availability of open-source code.

While LEDBAT design goals are sound, technical points have been raised by the scientific community participating to the LEDBAT working group, that ongoing discussion has not fully flattened yet. A legitimate question is whether adding LEDBAT to the already well populated world of Internet congestion control algorithms is really necessary and motivated. LEDBAT-reluctants suggest indeed to consider already existing, and therefore more stable and better understood, algorithms for low priority transport, such as NICE [81] or TCP-LP [60]. These comments, coupled with the move toward a closed source code, motivate the need for independent studies, so that claims concerning, e.g., the friendliness and efficiency of this new protocol, can be confirmed by the research community. This motivated the work carried out in this thesis.

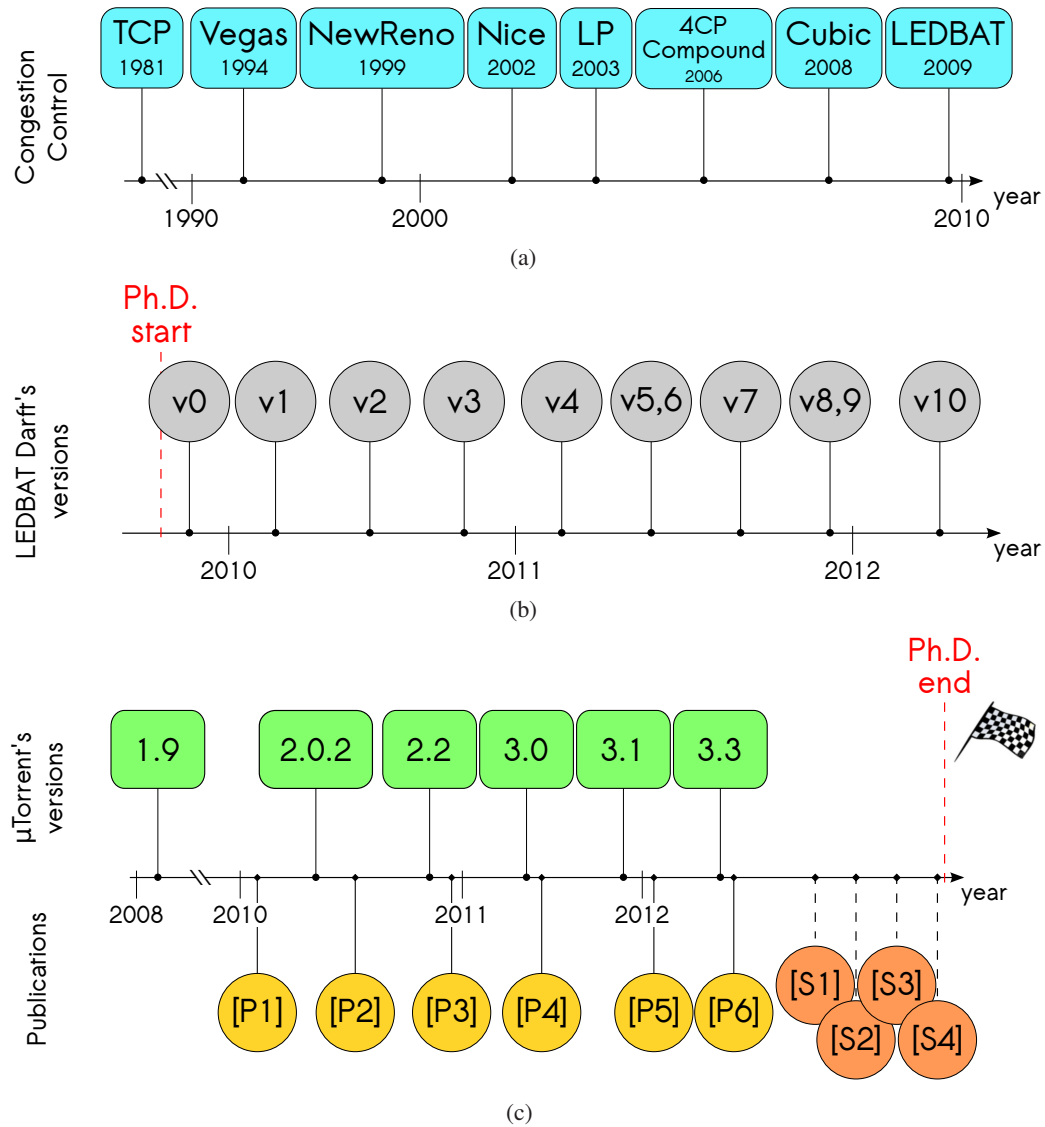


Figure 1.3: Timelines of the (a) relevant congestion control protocols, (b) LEDBAT's Draft specification versions, and (c) μ Torrent's client milestones and Publications (articles under review are represented with dashed lines).

1.2 The BitTorrent protocol in a nutshell

BitTorrent represents nowadays one of the most successful example of the P2P approach. It slowly took over a key service traditionally offered by the client-server paradigm: content distribution. The strength of this protocol lies in its scalability, its the ability to work across unreliable and heterogeneous networks, and its built-in incentive mechanisms, that encourage the data exchange among nodes.

Unlike traditional, client-server based, file-sharing applications, the main goal of BitTorrent is to setup an efficient system that involves a high number of hosts, called *peers*, which are either downloading the resource or uploading to others. To speed up content distribution and increase the number of content serving peers, the original resource is split in small fragments, or *chunks*, and then in smaller *blocks*. The *torrent file* is the metadata file that contains the index of all the chunks of a given content, as well as the fingerprint hash used to check the integrity of each chunk. To start the download, a peer has to retrieve the torrent file from a well known web-hosting site (*phase 1*). Using the information contained into the torrent file, the peer can contact a *tracker*, a centralized server that provides an initial list of neighbouring peers interested in the same resource (*phase 2*). This initial list is then periodically updated by the tracker itself and with the exchange of the lists from the neighbouring peers. Finally the peer can start to request the chunks to other peers involved in the swarm (*phase 3*). At the end the chunks are reassembled at the destination, once they all have been correctly downloaded. A simplified illustration of the aforementioned 3 phases is reported in Fig. 1.4.

In the following we provide an overview of the terminology used in the BitTorrent architecture and of the algorithms used to guarantee its proper functioning. Interested reader can refer to [3, 75] for additional details.

Terminology. We detail here the set of terms most commonly used when describing the BitTorrent data-sharing service.

- **Torrent.** A *torrent file* (or more often simply a *torrent*) is a resource descriptor of the content which is shared among peers. It contains a list of the chunks in which the resource is split, as well as fingerprint for each of them, used to validate the data integrity.
 - **Tracker.** A *tracker* is responsible of maintaining a list of all the peers involved in the content distribution, as well as collecting statistics on the downloading. The tracker is the only centralized element of the BitTorrent architecture.
 - **Seed and Leecher.** A peer which is still downloading the content from others is called *leecher*, while it becomes a *seed* as soon as it has downloaded all the content. The *initial seed* is the peer that owns the complete resource at the beginning of the sharing process.
 - **Swarm.** The *swarm* is the set of peers involved in the distribution of the same content.
-

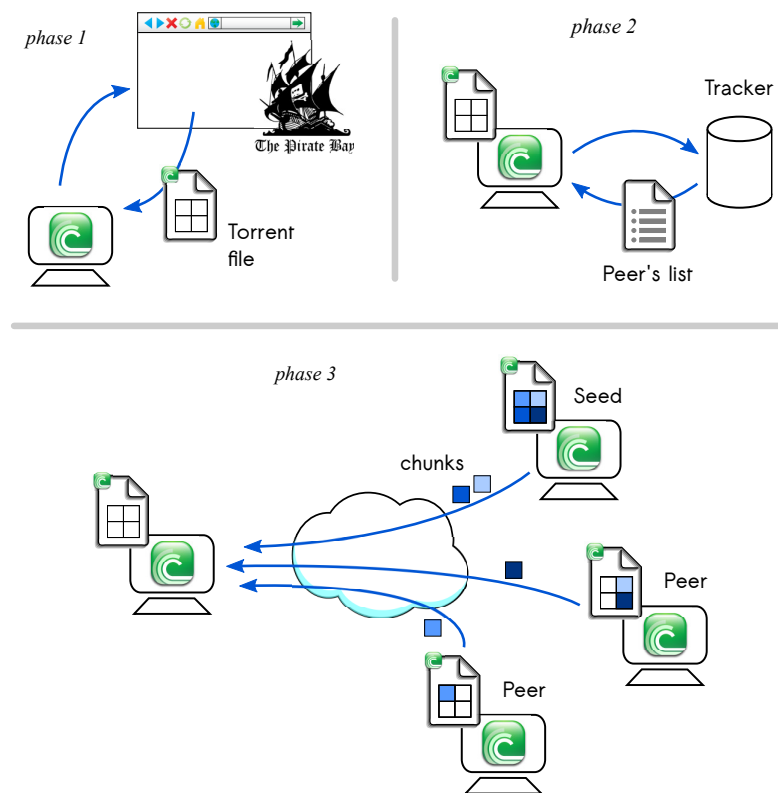


Figure 1.4: 3-steps diagram of BitTorrent functioning.

- **Chunk.** Before being transmitted, the original content is fragmented in several *chunks*. The size of each *chunk* depends on the content size, but usually takes value in the $[256 - 1024]$ KB range. Each chunk is then further divided into *blocks* of 16 KB.

Algorithms. In the following we describe the BitTorrent specific chunk-selection and peer-selection algorithms, aiming at encouraging fair trading and high availability of the resource, as well as efficient use of the available network capacity.

- **Pipelining.** The *pipelining* algorithm allows the peer to always have multiple pending requests (usually 5) at once.
- **Rarest-First.** The *rarest first* is a chunk-selection algorithm that ensures that rarest chunks among the neighbouring peers are downloaded first, while the more available ones are left for the end of the transfer.
- **Choking.** *Choking* is a temporary refusal to upload that is given to peers that do not participate to the data exchange and do not upload the chunks they have. The upload is accorded,

through an *unchoke* message, only to four remote peers from which we are downloading the most. This strategy is called *tit-for-tat* and is used to promote reciprocity and deter free-riding that often afflicts the P2P file-sharing services. So at each time, four connections are kept open using this choking/unchoking algorithm, while one connection is kept for the *optimistic unchoking*, which allows to explore the performance of other peers, by randomly select a peer to upload to.

- **Endgame mode.** Despite the use of rarest-first, some chunks may end up being less *globally* frequent than others. For this reason, near the end, the missing chunks are requested to all the neighbouring peers at the same time.
- **Seeding.** When in *seeding* mode, a seed performs the content's upload towards the peers that achieve the highest throughput. The goal of the algorithm is to maximize the transfer to the peers that can become seed faster.

Messages. The BitTorrent protocol uses a set of messages to coordinate the data dissemination and retrieval among the peers of the swarm. In this section we report on the messages most relevant to our analysis.

- **CHOKE and UNCHOKE.** The *CHOKE* message is sent to a peer to notify it that no request will be answered until the peer is *UNCHOKED*.
 - **HAVE.** The *HAVE* message is sent to the neighbouring peers upon completion of a given chunk.
 - **REQUEST.** When a peer wants to start the download of a chunk from a neighbour, a *REQUEST* message is sent out.
 - **PIECE.** The *PIECE* message is the one that contains the *block* of a requested chunk.
-

1.3 Related work

Literature effort relevant to this thesis is summarized in this section. For the sake of clarity and legibility, the articles are organized in three parts, covering the congestion control protocols in Sec. 1.3.1, Peer-2-Peer related work in Sec. 1.3.2 and recent LEDBAT publications in Sec. 1.3.3.

1.3.1 Congestion control

This section provides an overview of the relevant related work and background information about the Congestion Control (CC) protocols, with a special focus on *Lower-than Best Effort (LBE)* solutions. Congestion control studies on the Internet date back to [53] and it is out-of-scope to provide a full review of the existing literature here. Instead, we concentrate on a subset of such a solutions, considering four different categories of congestion control protocols, as presented in the taxonomy Fig. 1.5 and on a timeline in Fig. 1.3-(a). All the represented protocols in the taxonomy are TCP flavors, with the exception of LEDBAT which can be implemented on top of either TCP or UDP, as specified on [79].

We classify the protocols based on the *design strategy* to detect losses and the *aggressiveness* in capturing the available bandwidth. We consider two major *design strategies*: *loss based* and *delay based*. The loss based algorithms infer congestion on detecting a packet loss (by reception of duplicate acknowledgement or timeout expiration). The delay based protocols infer congestion in

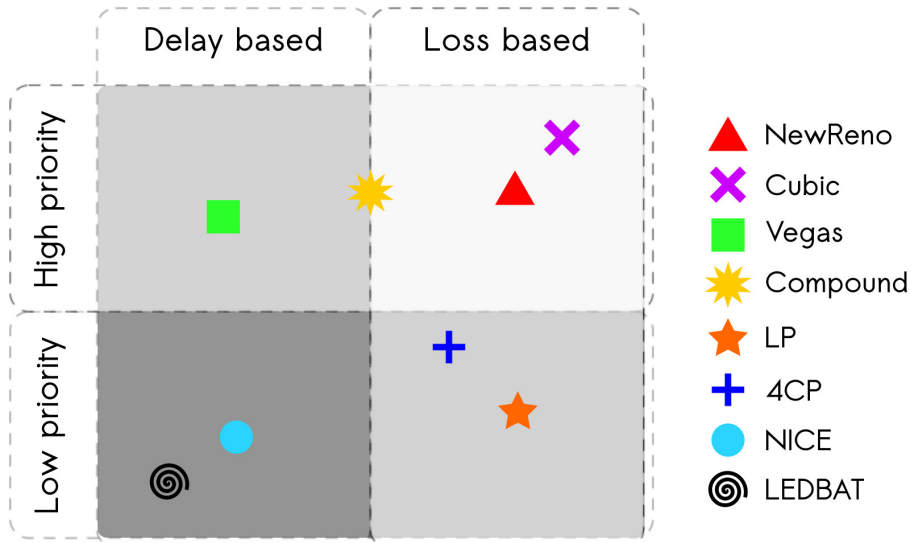


Figure 1.5: Congestion control design space: aim vs strategy. Our focus is on the protocols that combine *delay based* strategy with *low priority* goal.

the detected delay variation on the flow path. For each of the two design strategies, we consider two classes based on the aggressiveness in utilizing the end-to-end available bandwidth. We call these two classes *high priority* and *low priority*. The high priority algorithms are indeed very efficient (and aggressive) in using the spare bandwidth, while the low priority protocols are designed to use the spare bottleneck bandwidth as a scavenger service.

IETF endorses TCP NewReno [43], a high priority loss based congestion control algorithm for TCP. Recent evolutions of loss-based protocols include new algorithms like Cubic [77] and Compound TCP [80]. Cubic has become the default algorithm for TCP in the Linux since kernel version 2.6.18 and Compound TCP for the Windows operating system. 4CP [63] is a window based congestion control algorithm, implemented as a sender modification of standard TCP NewReno [43]. Its controller design exploits a bad phase (*congestion*) detector, in order to guarantee a long-term stable throughput to 4CP connection when competing with a single TCP flow, but use the available bandwidth when congestion is low. Two per-flow bandwidth guarantee configurations are possible: in fixed mode, the bandwidth is fixed by the user; in automatic mode, 4CP aims to be TCP-friendly over a large timescale. Low-priority loss-based congestion control has TCP-LP [60] as its best known example, which is also available as a Linux kernel module. TCP-LP [60] enhances the loss-based behavior of TCP Reno with an early congestion detection based on the distance of the instantaneous One-Way Delay from a weighted moving average calculated on all observations. In case of congestion, the protocol halves the rate and enters an inference phase, during which, if further congestion is detected, the congestion window is set to one and normal TCP Reno behavior is restarted. Vegas [27] was proposed as a high-priority delay-based congestion control alternative to the traditional loss-based TCP NewReno protocol. It derives its design choice of using the RTT delay measurements as a proactive congestion signal from the pioneering work of Jain about CARD [55], presented in late 80s. Like LEDBAT, Vegas aims at introducing a small fixed amount of additional delay in the bottleneck, yet to achieve a better throughput and reduce the retransmissions, as compared to standard TCP. Furthermore, the protocol assures fairness between multiple flows with heterogeneous propagation delays and does not suffer from stability problems. Among the low-priority protocols, NICE [81] extends the delay-based behavior typical of Vegas with a multiplicative decrease reaction to early congestion, which is actually detected when the number of packets experiencing a large delay in an RTT exceeds a given threshold. Overall the protocols closer in spirit with LEDBAT, i.e. which aim at implementing a LBE service for background transfers, are NICE and TCP-LP. Additional details about their similarities and differences are presented in Sec. 3.1.4.

Many more congestion control algorithms exist, tailored to the requirement of specific application and implemented at the application-layer, so to bypass OS modification [17, 18, 25, 35, 39, 57, 85]. More precisely, [57] adopts a L7 approach to infer the available capacity and tune the advertised receiver window size, in order to adjust the sending rate of the background service. Authors in [85] propose a solution which explicitly target the Peer-to-Peer world, with an application-level congestion control. Two components are used in this case: the first one which exploit throughput measurement to evaluate the available capacity of the access network, and a second element which set consequently with the previous results the number of simultaneous connections of the P2P

application. Finally [17] focuses on the YouTube application flow control, called *block sending*, using a passive analysis methodology over residential and academic network traffic. Especially their investigation unveils a detrimental effect of the control algorithm due to the interaction with TCP mechanism, particularly over congested links. In such conditions, almost half of the packet losses experienced by the user are due to large burst transmissions of packets, as an effect of the *block sending* sending policy. The black-box experimental measurements approach that we adopt in has been used in the past to unveil other proprietary algorithms for Peer-to-Peer services, as for Skype [25, 39] or P2P-TV applications [18, 35]. More precisely, authors in [25] investigate the user behavior, the codec selection, the signaling activity but also the traffic characteristics of the Skype service, which leverages also a P2P overlay. More precisely, they highlight a substantial difference in dealing with adverse network conditions (e.g. enforced path losses) when using TCP or UDP at the transport layer. Along similar lines [39] investigates how Skype Video behaves when is sharing the bottleneck with multiple other flows, in terms of efficiency and fairness, in a local testbed. From their analysis, authors found that in many cases the protocol is not able to fully utilize the available bandwidth, and in case of losses, the Skype Video flows are more aggressive than TCP. [35] contains an experimental analysis of two P2P-TV systems (PPLive and Joost) that focus on both the signaling process and content distribution mechanisms for the overlay network discovery and maintenance. A black-box approach is used also in [18], in order to analyse the reaction of P2P-TV applications under severe emulated network conditions. By means of `netem`, authors enforce impairments as the available bandwidth, latencies and losses, and discover that such systems are able to work also under adverse conditions, in some cases at the expense of the fairness to other TCP connections.

1.3.2 Peer-to-Peer applications

In this section we focus at related work on P2P applications. Specifically, we consider file-sharing (BitTorrent), VoIP (Skype) and P2P-TV (PPLive, SopCast, TV Ants, etc.) applications. A taxonomy of this research effort is proposed in Tab. 1.2.

Being the most successful Internet application for content distribution, BitTorrent has become over the years a rather popular research subject, with a multitude of publications aiming in delve its mechanisms, performance and behavior with manifold and complementary methodologies [20, 22, 23, 41, 52, 74, 75, 76, 86]. [75] is the pioneering work on BitTorrent and presents a fluid model for evaluating the stability, average download time of a single file and efficiency of the file-sharing architecture. Moreover, the authors investigated the built-in incentive mechanisms such as optimistic unchoke and peer-selection and validated the results with a discrete-event simulator and trace analysis from a real torrent swarm. Authors in [22] focused the attention on the BitTorrent performance, in terms of peers' link utilization and fairness of the volume of data served by each node, in a range of simulated scenarios. Their results confirm that the application is able to achieve nearly optimal uplink bandwidth utilization and download time. Furthermore, they propose some modifications to the tracker behavior and tit-for-tat policy to improve the data served fairness and avoid free-riding (which happen when a peer download data without contribute

Table 1.2: Taxonomy of the related P2P application research work.

Application	Reference	Approach	Comment
File-sharing (BitTorrent)	[75] Srikant et al.	Analytical model	performance evaluation, fluid model, incentive mechanism, game theory
	[22] Bharambe et al.	Simulation	link utilization, data exchange fairness, download time, tracker and tit-for-tat changes
	[23] Bindal et al.	Simulation	traffic locality, neighbor selection, tracker modification
	[41] Eger et al.	Simulation	flash crowd, constant population, packet- and flow-level simulation
	[52] Izal et al.	Measurement	single session and cluster performance, geographical analysis of peers
	[74] Pouwelse et al.	Measurement	content availability, integrity, flash-crowd handling, download performance
	[86] Zhang et al.	Measurement	discovery sites and multi-tracker crawling, geographical distribution, content analysis
	[20] Legout et al.	Experiments	incentive mechanism, clustering, tracker protocol extension, PlanetLab platform
VoIP (Skype)	[76] Rao et al.	Experiments	network latency, packet loss, torrent download time, controlled testbed
	[25] Bonfiglio et al.	Measurement	user classification, codec selection, adverse network impairments
P2P-TV (PPLive, SopCast, etc.)	[39] De Cicco et al.	Measurement	bandwidth efficiency, emulated network limitations, multi-flow fairness
	[35] Ciullo et al.	Measurement	signaling, content distribution, protocol adaptation to network impairments
	[18] Alessandria et al.	Measurement	available bandwidth, network latency, packet loss, peer selection

to the system). A simulative approach is used also in [23] to propose a new approach to improve BitTorrent traffic locality, by choosing a biased set of neighbors that could minimize the cross-ISP traffic in a real swarm. This kind of peer selection strategy is performed at the tracker level, so without the need of dedicated servers and without affecting the overall performance of the system. [41] proposes a comparison between flow-level and packet-level simulations, using their own `ns2` implementation, which is able to grasp the cross-layer interaction of the application and the transport layer. Authors use both real and star topology for the evaluation, as well as different scenarios including flash-crowd and constant peer population, and RTT heterogeneity among peers. We use this implementation of BitTorrent for `ns2`, available at [2], to perform our analysis in Chap. 6.

Studies as [52, 74, 86] use a measurement approach to shed light on the footprint in the Internet of BitTorrent and assess its influence in the gameplay of content distribution applications. In particular, [52] analyzes the log of a BitTorrent tracker, examining the scalability of the system, the flash-crowd effect in popularity and download speed of a single file, over a period of five months. Authors in [74] focused on the flash-crowd handling of the protocol, as well as the content avail-

ability and system integrity, while [86] aims in gaining a deep understanding of the file-sharing ecosystem by crawling five of the most popular torrent discovery sites for almost one year. Authors in [20, 76] also study BitTorrent download performance by means of either passive measurements or experimental tests, however they report on performance at a time when BitTorrent was using TCP, before the introduction of LEDBAT. More in details, [20] highlight the clustering process of peers with similar bandwidth properties and proved the effectiveness of the sharing incentives provided by the protocol. Moreover it points up the importance of a well provisioned initial seed capacity to achieve high overall system performance. Authors in [76] address the problem of reproducibility and reliability of experiment performed on dedicated clusters such as Grid'5000 [5]. We use the Grid'5000 platform in our experimental assessment presented in Chap. 5.

1.3.3 Recent LEDBAT work

At the same time, only few works have, for the time being, focused on LEDBAT aspects [15, 19, 28, 38, 59, 78]. In [38], BitTorrent developers detail a specific aspect of their implementation: namely, an algorithm to solve the problem of the clock drift in LEDBAT, to ameliorate the queuing delay estimation at the sender side. Authors in [78] instead study LEDBAT in a local testbed, employing different real ADSL modems, focusing on the interaction of LEDBAT and active queue management techniques that are becoming commonplace in modern home gateways. As a result, the use of such mechanism could allow a finer tuning of priorities among different flows, but in while the interaction between AQM and LEDBAT could results in poor performance.

An investigation on the policies for a runtime parameter tuning is presented in [15]. In particular, the author propose a modification to the algorithm to dynamically set the *gain* value, without affecting the original goals of LEDBAT. Instead in [19] authors evaluate a Python user-level implementation of the new protocol over an emulated network and in a large testbed. Their results confirm that LEDBAT is able to operate without interfering with the interactive flows, while an efficiency problem arise in exploiting all the available network capacity, which can be due the user-space implementation used for the study.

[28] suggests four different solutions to the unfairness issue that affects the LEDBAT protocol: (i) random pacing of packets in a RTT, (ii) slow-start, (iii) random congestion window drop and (iv) simple multiplicative decrease scheme. The proposed solutions come with a performance trade-off between fairness and utilization of the network resources. Authors in [59] present an evaluation of different schemes for the decrease phase of the LEDBAT congestion window, which include the initial linear controller, and a multiplicative decrease policy. Nevertheless their solution is focused on achieving an high link utilization, regardless the intra-protocol fairness property.

1.4 Contributions of this thesis

The goals that this thesis aims to achieve are:

- To shed light on the inner mechanisms of LEDBAT and its design choices.
- To understand the behavior of the main BitTorrent client implementation in a controlled environment, under a wide range of scenarios and forced impairments.
- To quantify the level of low-priority of LEDBAT with respect to other LBE congestion control and the tunability of its behavior, through the main working parameters.
- To prove and correct the flaws of the protocol highlighted by analytic model, simulations and measurement experiments.
- To quantify the performance impact over TCP flows, both in simulated scenarios and real-world case.
- To evaluate the impact of the LEDBAT on the BitTorrent application, both in simulation and real networks.

Tab. 1.3 contains an overview of the contributions of the thesis.

Table 1.3: Synopsis of the thesis.

	Chapter	Methodology	Comment	Publication
Part. I	Chap. 2	Experimental testbed	Measurement, reverse engineering, black-box approach	[P1]
	Chap. 3	Simulative approach	Performance assessment, sensitivity analysis, comparison with other LBE protocols	[P2, P3]
	Chap. 4	Simulative approach, Math. model	Unfairness issue, proposed solution fLEDBAT	[S1]
Part. II	Chap. 5	Simulative approach	Impact of LEDBAT on BitTorrent performance	[P4]
	Chap. 6	Experimental approach		[P5]

The first part of this thesis is dedicated to the analysis of the LEDBAT protocol under the congestion control point of view. To this aim, different methodologies are exploited, as (i) experimental testbed, (ii) simulation programs, and (iii) analytical model. In order to compare the performance of the novel protocol with respect to standard TCP or other LBE solutions, different flow-centric metrics are used, as the intra-protocol and inter-protocol fairness, the network efficiency, the buffer occupation, to cite a few.

In Chap. 2, we use an active testbed to study different flavors of the LEDBAT protocol, which correspond to different milestones in the BitTorrent software evolution. Notice that, at the time this study was made, no publicly description of LEDBAT not uTP was available. Thus, the black-box

approach was the only feasible methodology. Focusing on single flow scenario, we investigate emulated artificial network conditions, such as additional delay and capacity limitation. Then, in order to better grasp the potential impact of LEDBAT on the current Internet traffic, we consider a multiple flow scenario, and investigate the performance of a mixture of TCP and LEDBAT flows, so to better assess what *Lower-than Best Effort (LBE)* means in practice.

In Chap. 3 we have an initial discussion on the LEDBAT IETF draft, which describes the operations of the novel congestion control algorithm. Then we use our `ns2` implementation of the protocol to evaluate the performances via packet-level simulation. We found that the new protocol successfully meets some of its design goals, as for instance the efficiency in exploiting the available network capacity. Then, we focus on a comparison with other LBE protocols such as TCP-LP and NICE. The aim of such investigation is indeed to quantify and relatively weight the level of low-priority provided by such protocols. Moreover, a careful sensitivity analysis on the LEDBAT performance is carried out, by tuning its main parameters in both an inter- (against TCP) and intra-protocol (against LEDBAT itself) scenarios.

In Chap. 4 we unveil that LEDBAT is affected by a latecomer advantage, where newly arriving connections can starve already existing flows. For this reason, we introduce `fLEDBAT` (that stands for fair-LEDBAT), that modifies the original LEDBAT design to solve this issue, while preserving the intra-protocol fairness and network efficiency. A fluid model of the protocol is used to provide closed-form expression for the stationary throughput and queue occupancy in simple scenarios. Finally a more realistic scenario, which use a P2P-like traffic model, with heterogeneous Round Trip Times and web background traffic, is evaluated.

The second part of the thesis is focused on LEDBAT as a major player in the P2P ecosystem, so we study its influence on the BitTorrent performance, both via simulations and using real network experiments.

In Chap. 5 we assess the impact of LEDBAT on the primary BitTorrent user-centric metric, namely the torrent download time, by means of packet level simulation. In this case we use the BitTorrent implementation [2] with our LEDBAT implementation [7] for `ns2`. The study is performed under different conditions that involve homogeneous (all peers use the same protocol for data exchange) or heterogeneous (mixed use of TCP and LEDBAT protocol) swarm population. Surprisingly we find out that LEDBAT swarm achieve a lower completion time with respect to TCP swarm. Then, we consider a scenario in which peers within the same swarm use an heterogeneous target delay setting, similarly to the sensitivity analysis carried out with a flow point of view.

On the same line, Chap. 6 present a study on the impact of LEDBAT on the torrent completion time, but this time using a experimental approach over a dedicated cluster platform, namely the Grid'5000 as in [76]. As in the previous chapter, the scenario consider both homogeneous and heterogeneous swarm population, and asses the performance achieved with the default settings of the BitTorrent client, which yield to use a specific mixture of TCP and LEDBAT protocol for data transmission. In the last part of the chapter we investigate on the connection between data and control plane when providing a timely content distribution.

Finally in Chap. 7 we discuss the main results achieved in this thesis, with a glance on the future directions and evolutions of this study.

Part I

Congestion control perspective

Chapter 2

Performance evaluation - Black box experiments

In this chapter, we present an initial set of findings on the LEDBAT protocol, gathered using a black-box approach on the BitTorrent client on a controlled local testbed.

The aim of this analysis is twofold. On the one hand, we target at understanding the performance of LEDBAT in a number of simple single flow scenarios, considering multiple versions of the official client so to better clutch its evolution and investigating their behavior in emulated artificial network conditions. On the other hand, by means of multiple flows scenarios, we aim at gathering a preliminary understanding of the implication that a widespread adoption of LEDBAT could have on the current Internet landscape and to assess what LBE means in practice. We tackle the above issues with an active-measurement black-box study of the official BitTorrent client, since the analysis was performed before the release of the protocol specification [79]. Although the investigation by simulation carried out in Chap. 3 brings new light to the protocol understanding, we find active testbed experimentation extremely useful for several reasons. First, the BitTorrent implementation of the LEDBAT protocol may differ from any draft-compliant implementation by some design choices or parameter setting, that may have a deep impact on the protocol performance. Second, the most widespread LEDBAT implementation on the Internet will be the official BitTorrent version, rather than a legacy implementation, which motivates a direct evaluation of this client. Third, from our point of view, the analysis of proprietary applications by independent observers has the benefit of shedding light on the protocol inner workings. Finally real-world dynamics introduced by network devices are often much more complex than the synthetic ones that a simulation environment, although accurate, can reproduce.

The remainder of this chapter is organized as follows. In Sec. 2.1 we describe the methodology and present the initial findings on the client evolution over time. The results of the experiments performed in single-flow scenarios are presented in Sec. 2.2, while in Sec. 2.3 we report the outcome of multiple-flow experiments, carried out either on our active testbed or over an ADSL link. Lastly in Sec. 2.4 we summarize the relevant findings gathered with this approach.

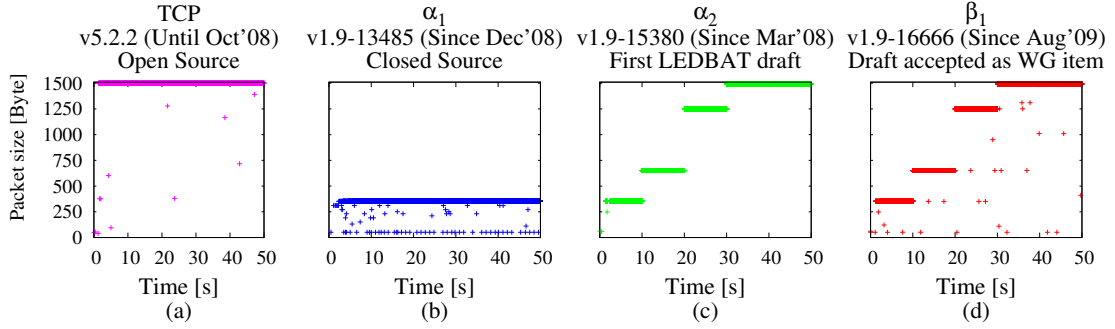


Figure 2.1: The last few months of BitTorrent client evolution: Temporal plot of packet-level traces for different BitTorrent flavors, reporting packet size during the first minute of the transfer.

2.1 Methodology and Preliminary Insights

For the investigation of the LEDBAT, we resort an active measurements experimental approach, consisting in the analysis of the traffic generated by the BitTorrent client on different network scenarios. We run several versions of the new BitTorrent client on PCs equipped with dual-core processors featuring (i) unless otherwise stated, native installations of Windows XP or (ii) BitTorrent clients running on Linux using the `wine` Windows emulator. PCs are either (i) connected to the Internet through ISPs offering ADSL access, or (ii) in a local LAN testbed via Ethernet cards. In the first case we leave the default modem settings unchanged, while in the second one we disable the interrupt coalescing feature and avoid the usage of jumbo frames. Moreover in the LAN testbed, the traffic is routed through a middlebox running a 2.6.28 Linux kernel, which acts also as network emulator by means of `netem`, in order to enforce artificial network conditions.

As formerly stated, in our experiments we consider both single flow and multiple flows scenarios. Single flow experiments are useful to understand the protocol performance under a range of different network conditions, while multiple flows experiments are needed to quantify the level of inter-protocol priority (e.g., with respect to TCP flows) and intra-protocol fairness (e.g., with respect to other LEDBAT flows) achieved by the distributed control algorithm. Under the classic BitTorrent terminology, every LEDBAT sender-receiver pair is a seeder-leecher pair, so that data transfer happens in a single direction. In case of multiple-flows experiments, every pair of actors belongs to a different torrent, so that no data exchange happens between different leechers.

We start by providing some insights on the BitTorrent evolution with the help of Fig. 2.1. Every picture refers to a different experiment, of which we report the first minute, corresponding to a different BitTorrent flavor. The seeder connects to the middlebox with a 100 Mbps Ethernet link, while between the middlebox and the leecher there is a 10 Mbps Ethernet bottleneck link. No other traffic is present on the bottleneck, and the one-way delay on the forward path is forced to 50 ms, to loosely emulate a scenario where two faraway peers with high speed Internet access (e.g.,

ADSL2+, FTTH or Ethernet) are connected together.

Pictures are arranged so that the macroscopic timescale of BitTorrent evolution also grows from left to right: Fig. 2.1-(a) shows, as a reference, the old open-source TCP-based client, while Fig. 2.1-(b) refers to the first closed-source version α_1 , released December 2008. Then, Fig. 2.1-(c) depicts the α_2 version, released roughly at the same time of the first IETF Draft [79] in March 2009. Finally, Fig. 2.1-(d) refers to the β_1 version, released after the Draft was accepted as an official IETF WG item in August 2009.

The comparison of different versions of the protocol yields several interesting observations. First, notice that all versions analyzed correspond to important milestones in the development process of the protocol: thus, they provide a valuable perspective which highlights the flaws as well as the improvements of the subsequent steps of LEDBAT evolution. In particular, the α_1 version (which precedes the Draft specification and motivates a black-box approach) was particularly unstable and soon superseded. Moreover, from this study it emerges that the LEDBAT implementation is *constantly* evolving: as such, we believe that picking a single version, such as the most recent one, would limit the scope of our study.

For each flavor represented in Fig. 2.1, pictures depict the packet size on the y-axis, measured at the sender side, with time of the experiment running on the x-axis. As it can be seen, the application-layer segmentation policy is remarkably variable across different LEDBAT flavors. In contrast with TCP, which always transmits segments of maximum size, LEDBAT instead uses variable packet sizes. For instance, the α_1 implementation of Fig. 2.1-(b) mostly used small segments of about 350 Bytes, transmitted at very high rate. Although this allows a finer tuning of the congestion window size, (e.g., likely to be more reactive to network condition), it definitively results in an unnecessary overhead. This segmentation policy is a bad choice for large transfers, and was indeed soon dropped in favor of larger segment sizes. As can be gathered from Fig. 2.1-(c) and Fig. 2.1-(d), newer BitTorrent flavors start by segmenting data in small-size segments, and then gradually increase the segment size over time, rarely changing it once the full-payload segment size is reached. In case of α_2 flavor, we observe subsequent phases, about 10-seconds long, where only a single segment size is used: it takes about 40 seconds to the application-layer segmentation policy to settle to full-payload segment size. The β_1 flavor behaves similarly, although a wider range of segment sizes is employed during the whole experiment, probably to obtain a finer byte-wise control of the congestion window.

The corresponding time evolution of the achieved throughput, measured over 1 s time-windows is depicted in Fig. 2.2-(a), using a longer timeframe of about 4 minutes. We merely superpose the curves for the sake of comparison, but experiments have been independently performed. It can be seen that, shortly after achieving a sustained throughput of about 9 Mbps during about 50 seconds, the sending rate of the α_1 version suddenly drops, and about 2 minutes are necessary to recover from this starvation (this unstable behavior was observed under a wide range of conditions). In contrast, α_2 and β_1 achieve a lower but steady throughput, slightly above 4 and 7 Mbps respectively.

As a reference, we also report the throughput of a BitTorrent client using TCP running on the native Windows and Linux networking stacks under their default settings. The networking stack implementation and configuration dramatically impacts the protocol performance also in the TCP

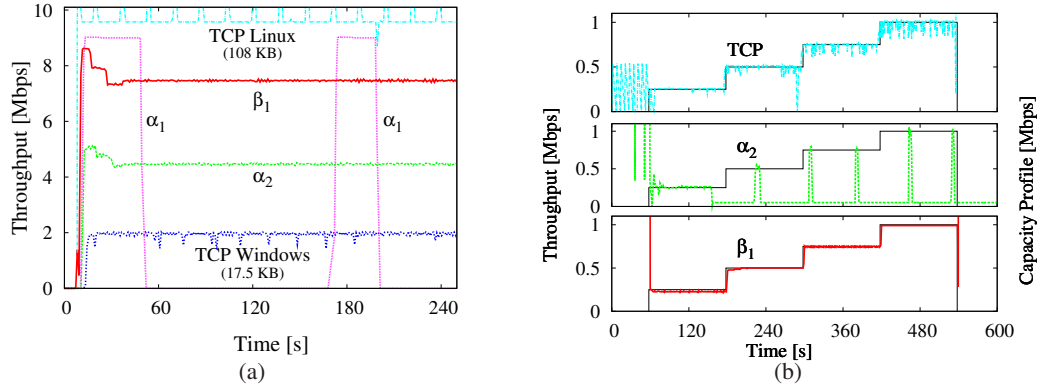


Figure 2.2: Throughput for different flavors (a) without and (b) with bottleneck capacity limitations.

case. As reported in [29], in Windows XP, for transmission rates between 10-100 Mbps the default receive window is set to 17520 Bytes, whereas the default value of the Linux receive window (set in `net.ipv4.tcp_mem`) is about 6 times larger. Notice that in the Windows XP case, due to the 50 ms delay, the default value of the maximum window is not large enough to allow full saturation of the bottleneck pipe. This is an important, though not novel, observation on which we will come back later on Sec. 2.3.

2.2 Single flow scenario

In this section, we start with simple single flow scenarios so to refine the performance pictures of the different flavors by testing the impact of varying network conditions. Among the several experiments conducted, we report the most relevant for our performance evaluation. In more detail, we consider (i) bottleneck capacity limitations, (ii) one-way delay impairment on either the forward or the backward path and (iii) different access technologies.

Let us start by testing how BitTorrent copes with changing bottleneck capacity. We use a setup similar to the former experiment, but in this case the capacity of the link between the middlebox and the leecher is limited by means of the Hierarchical Token Bucket (HTB), available in `netem`. In more detail, we start at $t=60$ s to let LEDBAT throughput settle to a steady state, and then we turn on the HTB shaper. We initially tune it to 250 Kbps, increasing then the available capacity in steps of 250 Kbps every 2 minutes, as shown by the solid line capacity profile in Fig. 2.2-(b). A decreasing capacity profile yields to similar results and is thus not shown in the figure.

Time evolution of the throughput is reported for the new α_2 , β_1 flavors as well as for the old TCP client. Flavor α_2 proves to be unable to quickly adapt to the changing link rate: it periodically enters a probing (or slow-start) phase, where it likely tries to infer network conditions by varying the segment size and sending rate. However, this phase is apparently unsuccessful and α_2

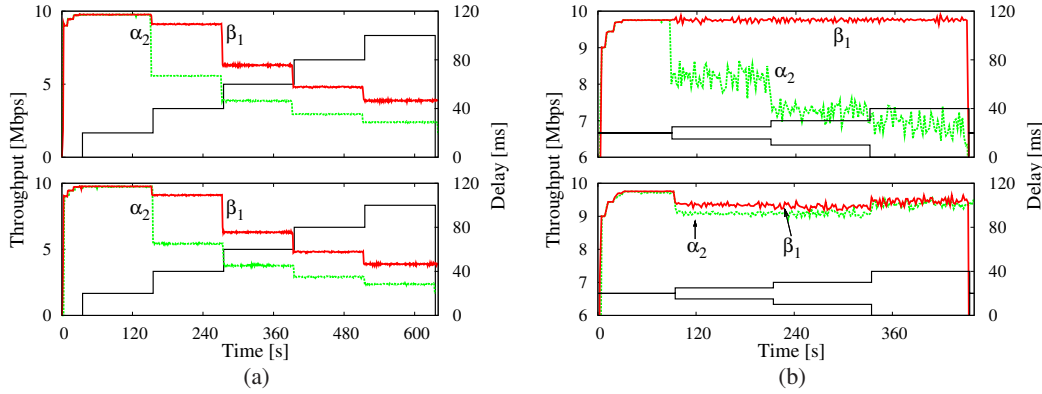


Figure 2.3: Throughput evolution for different delay settings on the forward (top) and backward (bottom) path: (a) average delay increases over time, delay is equal for all packets (b) average delay is constant over time, delay variance increases over time.

throughput starves (we did not observe such a starvation phenomenon for bottleneck larger than 1000 Kbps). This bug has been fixed by later releases: β_1 matches the available bandwidth, and moreover LEDBAT shows a much smoother curve than TCP. In this case, we may say that one of the LEDBAT design goals, namely, to efficiently exploit the available capacity, seems to be perfectly achieved.

Then, consider that the LEDBAT congestion control is based on a linear adaptation (i.e., growth/shrink) of the sender window to variations in the queuing delay on the forward data path (i.e., as inferred by the decrease/increase of the one-way delay, with respect to the minimum measured one as reference): it is thus critical to assess its reaction to the measured One-Way Delay (OWD). However, the sender response to queuing delay variations is nevertheless based on a closed-loop reaction with the receiver: therefore, we argue that the time instants at which the sender window growth/shrink decisions will be taken are also affected by the two-way delay, or Round Trip Time (RTT).

Thus, we setup and experiment in which we add an incremental OWD on either the forward (data) or backward (acknowledgement) paths. As before, after LEDBAT settles we increase the additional delay in steps of 20 ms every 2 minutes, for an RTT spanning on the 20–100 ms range as shown by the stepwise profile in Fig. 2.3-(a). The amount of OWD delay is added either to the forward path (top) or backward (bottom) path: in the former case, the delay incrementally adds to the OWD estimation performed by the sender so that it may directly affect the congestion control loop, while in the latter case it only delays the acknowledgement and may only indirectly affect the control loop.

As it can be seen from the comparison of the top and bottom plots of Fig. 2.3-(a), the overall effect on performance is the same: BitTorrent throughput decreases for increasing RTT, which is

due to an upper bound of the receiver window (analogously to what seen before for TCP). With some back-of-the-envelope calculation based on the experimental results shown in Fig. 2.3-(a), one can gather that the receiver window limit has been increased from 20 full-payload segments of α_2 to 30 full-payload segment of β_1 . While the picture shows that this limit may not be enough to fully utilize the link capacity (e.g., β_1 achieves about 4 Mbps throughput on a 10 Mbps link with $\text{RTT}=100$ ms), in practice it is not a severe constraint, as the capacity will likely be shared across several flows established with multiple peers of a BitTorrent swarm (or the receiver window limit could be increased).

In Fig. 2.3-(b) we instead investigate the effects of a variable OWD delay, that changes for each packet uniformly at random, with average OWD equal to 20 ms. In this case we keep the average constant but increase the delay *variance* every 2 minutes, so that the profile reports the minimum and maximum delays of the uniform distribution. The variable delay also implies that packet order is not guaranteed, because packets encountering a larger delay will be received later and thus out-of-order. Again, delay variance is enforced on either the forward (top) or backward (bottom) path. As it can be expected, LEDBAT is rather robust to a variable jitter on the backward path, where we observe only a minimal throughput reduction. Conversely, variance in the forward path has a much more pronounced performance impact: interestingly, α_2 throughput significantly drops, whereas β_1 performance is practically unchanged. This probably hints to the use of a more sophisticated noise filtering algorithm (e.g., that discards delay samples of out-of-order packets), although a more careful analysis is needed to support this assertion.

We finally perform an experiment using PCs connected through ADSL modems to the wild Internet. Thus, in this case we no longer have complete control over the network environment, but we still can assume that no congestion happens in the network and that the access link constitutes the capacity bottleneck. It can be seen from Fig. 2.4-(a) that in a realistic scenario, when the end-hosts only run LEDBAT, β_1 achieves a smooth throughput whose absolute value closely matches the nominal ADSL uplink capacity (640 Kbps). In contrast, TCP throughput is more fluctuating due to self-induced congestion, which causes fairly large queues before eventual losses occur. This confirms that the goal of avoiding self-induced congestion at the access is also met.

2.3 Multiple flows scenario

We now consider a scenario in which (i) a single TCP flow interferes with LEDBAT on either the forward or the backward path, and then (ii) multiple flows share the same bottleneck link, varying the ratio of LEDBAT and TCP flows so to better assess the protocols mutual influence.

We now explore scenarios with several concurrent flows, starting with the simple one where a single LEDBAT flow interacts with a single TCP flow. Considering two PCs connected through ADSL modems to the wild Internet, Fig. 2.4-(b) reports an experiment where, during a single LEDBAT transfer, we alternate periods in which PCs generate no traffic other than LEDBAT, to periods (i.e., the gray ones) in which we superpose TCP traffic on either the forward or backward path.

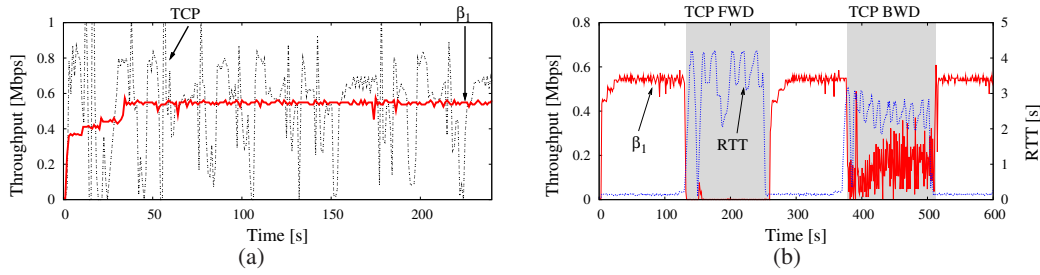


Figure 2.4: Real Internet experiments: (a) different flavors and (b) interfering traffic.

The plot reports the time evolution of the LEDBAT throughput as well as the RTT delay measured by ICMP (as a rough estimation of the queue size seen by LEDBAT). During the silence periods (0–120 s and 240–360 s), as bottleneck is placed at the edge of the network, LEDBAT is able to efficiently exploit the link rate. As soon as a backlogged TCP transfer is started on the forward path (120–240 s), LEDBAT congestion control correctly puts the traffic in low priority. Notice that in this case, ICMP reports that a fairly large queue of TCP data packets builds up in the ADSL line (roughly 4 seconds, corresponding to about 300 KB of buffer space for the nominal ADSL rate). Conversely, whenever the backlogged TCP transfer is started on the backward path (360–480 s), LEDBAT transfer on the forward direction should only be minimally affected by the amount of acknowledgement TCP traffic flowing in the forward direction. However, as it can be seen from Fig. 2.4-(b), the LEDBAT throughput drastically drops, further exhibiting very wide fluctuations (notice also that the ADSL modem buffer space of the receiver appears to be smaller, as the RTT is shorter). Notice that in this case, LEDBAT forward data path shares the link capacity only with TCP acknowledgements, which account for a very low, but likely very bursty, throughput: this may lead LEDBAT into a messy queuing delay estimate, and as a result, the uplink capacity of the device is heavily underutilized (about 74% of wasted resources).

We finally perform experiments to analyze the interaction of several flows. In this case, we setup several torrents, one for every different LEDBAT seeder-leecher pair, so that no data exchange happens between leechers of different pairs. Thus, flows are independent at the application layer, though they are dependent at the transport layer, as they share the same physical 10 Mbps RTT=50 ms bottleneck.

We consider a fixed number of $F=4$ flows, and vary the number of TCP and LEDBAT β_1 connections to explore their mutual influence. All flows start at time $t = 0$, experiments last 10 minutes and results refer to the last 9 minutes of the experiment. We generate TCP traffic using Linux (so that we can reliably gather retransmission statistics using `netstat`), setting the congestion control flavor to TCP NewReno. We perform two set of experiments, using either the Windows or Linux defaults values for the maximum receiver windows as early stressed in Fig. 2.2-(a): in our setup, the Windows-like TCP settings (TCP_W) are thus less aggressive than Linux ones (TCP_L).

Table 2.1: Efficiency and Fairness between multiple TCP and LEDBAT flows

		TCP _W , LEDBAT β_1						
TCP	LEDBAT	% ₁	% ₂	% ₃	% ₄	η	Fairness	RTX%
4	0	0.25	0.25	0.25	0.25	0.67	1.00	5e-4
3	1	0.14	0.14	0.14	0.57	0.94	0.64	-
2	2	0.10	0.10	0.40	0.40	0.93	0.74	-
1	3	0.08	0.31	0.31	0.31	0.92	0.87	-
0	4	0.25	0.27	0.24	0.24	0.96	1.00	-

		TCP _L , LEDBAT β_1						
TCP	LEDBAT	% ₁	% ₂	% ₃	% ₄	η	Fairness	RTX%
4	0	0.25	0.25	0.25	0.25	0.98	1.00	0.06
3	1	0.35	0.32	0.32	0.00	0.98	0.75	0.14
2	2	0.43	0.51	0.03	0.03	0.98	0.56	4e-3
1	3	0.87	0.04	0.04	0.05	0.98	0.33	-
0	4	0.25	0.27	0.24	0.24	0.96	1.00	-

For each experiment, we evaluate user-centric performance by means of the breakdown of the resources acquired by each flow, while we express network-centric performance in terms of the link utilization η . To further quantify the protocol mutual influence, we use the Jain’s fairness index of the flows throughput and evaluate the percentage of TCP retransmissions (RTX). Results are reported in Tab. 2.1, with Windows and Linux settings on the left and right respectively. Comparing the two table portions, we argue that the exact meaning of “low-priority” may be fuzzy in the real-world. Indeed, while LEDBAT β_1 is lower priority than an “aggressive” TCP, it may be competing more fairly against a more gentle set of parameters, thus being at least as high priority as TCP. In fact while LEDBAT is practically starved by TCP_L, LEDBAT is able to achieve a slightly higher priority than TCP_W.

Although we recognize that results may change using more realistic and heterogeneous network scenarios, or using the real Windows stack instead of simply emulating its settings, we believe that an important point remains open: i.e., the precise meaning of “lower than best effort”, as the mutual influence of TCP and LEDBAT traffic may significantly differ depending on the TCP flavor as well.

2.4 Summary

In this chapter we presented an experimental evaluation of LEDBAT, the novel BitTorrent congestion control protocol. Single-flow experiments in a controlled environment show some of the fallacies of earlier LEDBAT flavors (e.g., instability, small packets overkill, starvation at low throughput, tuning of maximum receiver windows, wrong estimate of one-way delay in case of packet reordering, etc.), that have been addressed by the latest release. Experiments in a real Internet environment, instead, show that, although LEDBAT seems a promising protocol (e.g., achieving a much smoother throughput and keeping thus the delay on the link low), some issues still need to be worked out (e.g., performance in case of reverse path traffic). Finally, multiple-flows experiments show that “low-priority” meaning significantly varies depending on the TCP settings as well.

A first step toward the analysis of LEDBAT performance is made in this chapter. Indeed more effort is indeed needed to build a full relief picture of the LEDBAT impact on other interactive applications (e.g., VoIP, gaming), explicitly taking into account the QoE resulting from their interaction. Also, the methodology could be refined by, e.g., instrumenting the Linux kernel to measure the queue size, or by inferring the OWD measured by LEDBAT by sniffing traffic at both the sender and receiver, etc. Finally, the boundaries of the investigation could be widened by taking into account the effects of LEDBAT adoption on the BitTorrent P2P system itself, as for instance LEDBAT interaction with throughput based peer-selection mechanism, or its impact on files download time, as we dig in Chap. 6.

Chapter 3

Performance evaluation - Simulation based

In this chapter, we perform an in-depth study of LEDBAT, after its specifications were published in late 2009 with an IETF Draft [79]. Thus, in Sec. 3.1, we first describe the LEDBAT algorithm as it is presented in the Draft, focusing on its design choices and delineating the goals and objectives it aims to achieve. Then, we implement the novel congestion control algorithm and investigate its performance by means of packet-level simulations, which are presented in Sec. 3.2. Considering a simple bottleneck scenario, where the new LEDBAT competes against either TCP or other LEDBAT flows, we evaluate the fairness of resource share as well as the protocol efficiency. Our initial results show that the new protocol successfully meets some of its design goals, as for instance the efficiency one. Moreover it correctly implements a *Lower-than Best Effort (LBE)* service, as the LEDBAT flows is proven to be non-intrusive when competing against TCP in homogeneous conditions.

In the second part of the simulations section, we focus on a performance comparison with other LBE protocols, such as TCP-LP and NICE, in order to quantify and relatively weight the level of aggressiveness provided by such protocols. Our results show that LEDBAT transport generally achieves the lowest possible level of priority, with the default configurations of NICE and TCP-LP representing increasing levels of aggressiveness. In addition, we perform a careful sensitivity analysis of LEDBAT performance, by tuning its main parameters in both an inter-protocol (against TCP) and intra-protocol (against LEDBAT itself) scenarios. In the inter-protocol case, even in case of misconfiguration LEDBAT competes as aggressively as TCP, but we show that it is not possible to achieve an arbitrary level of low-priority by merely tuning its parameters. In the intra-protocol case, we show that coexistence of legacy flows with slightly dissimilar settings, or experiencing different network conditions, can result in significant unfairness. Finally we conclude the chapter in Sec. 3.3.

3.1 LEDBAT overview

This section provides a basic overview of the LEDBAT Draft [79]. To better understand the motivations behind LEDBAT, let us recall that the standard TCP congestion control needs losses to back off: this means that, under a drop-tail FIFO queuing discipline, TCP necessarily fills the buffer. As uplink devices of low-capacity home access networks can buffer up to hundreds of milliseconds, this may translate into poor performance of interactive applications (e.g., slow Web browsing and bad gaming/VoIP quality).

To avoid this drawback, LEDBAT implements a distributed congestion control mechanism, tailored for the transport of non-interactive traffic with LBE (i.e., lower than TCP) priority, whose main design goals are:

- Saturate the bottleneck when no other traffic is present, but quickly yield to TCP and other UDP real-time traffic sharing the same bottleneck queue.
- Keep delay low when no other traffic is present, and add little to the queuing delays induced by TCP traffic.
- Operate well in drop-tail FIFO networks, but use explicit congestion notification (e.g., ECN) where available.

Intuitively, to saturate the bottleneck it is necessary that queue builds up: otherwise, when the queue is empty, at least sometimes no data is being transmitted and the link is under-exploited. At the same time, in order to operate friendly toward interactive applications, the queuing delay needs to be as low as possible: LEDBAT is therefore designed to introduce a non-zero *target* queuing delay.

In order to achieve this goal, LEDBAT follows a simple strategy. First of all, it exploits the ongoing data transfer to Measure the One-Way Delay (OWD), from which it derives an estimate of the *queuing delay* on the forward path. Using OWD instead of Round Trip Time has the main advantage of preventing unrelated traffic on the backward path from interfering with data transmission. Second, it employs a *linear controller* to modulate the congestion window, and consequently the sending rate, according to the measured delay. LEDBAT operations can be summarized in the pseudocode in Fig. 3.1.

In the following, we first consider the two main components of the LEDBAT algorithm separately, and then we report some further considerations on the TCP-friendliness of the novel protocol. Finally we detail the common aspects and main differences of the novel protocol with respect to existing LBE proposals.

3.1.1 Queuing Delay Estimate

Delay measurements are performed collaboratively by the sender and the receiver. The former puts a timestamp from its local clock in each packet. The latter, instead, calculates the One-Way Delay as the difference between its own local clock and the received timestamp, and communicates it

back to the sender in the acknowledgements. The sender, besides, maintains a minimum of all observed delays, which represents the *base delay* used in queuing delay estimate.

To explain the rationale behind such technique, let us consider the different components of OWD: propagation, transmission, processing and queuing. Neglecting the processing delay, propagation and transmission delays are constant components, while the only variable component is the queuing delay. Intuitively, a packet which finds the queue empty (i.e., null queuing delay) will accurately estimate the constant portion of the OWD (i.e., the sum of propagation and transmission delays). This measure yields a minimum of the delay, that will be stored as a reference: then, the queuing delay can be estimated as the difference between the current and the reference delay.

One-way delay measurements are notoriously difficult, especially for non-synchronized hosts. Yet the *variation* of delay with respect to the base delay, which is actually exploited by LEDBAT, is a much more robust metric. In particular, it does not suffer from timestamp errors such as fixed offsets and skews from the true time. For instance, the sender and receiver offsets could severely affect the absolute OWD estimate, but they happily cancel in the arithmetic difference $\text{queuing_delay} = \text{current_delay} - \text{base_delay}$ (since both delays correspond in their turn to the difference of the receiver minus the sender delay). Further considerations about clock skew, noise filtering and route changes issues can be found in [79].

3.1.2 Controller Dynamics

A *proportional-integral-derivative (PID)* controller governs the dynamic of the congestion window in both the ramp-up and ramp-down phases. The controller continuously adapts the window to the estimated delay, in order to match the target delay. Clearly, when the queuing delay estimate is lower than the target (i.e., $\text{off_target} < 0$) the sending rate has to increase, so that queuing delay reaches the target. Conversely, when the queuing delay estimate is higher than the target (i.e., $\text{off_target} > 0$) the controller slows down the sending rate.

In Fig. 3.1 we observe that the controller itself is characterized by two parameters, the *TARGET* delay and the *GAIN* coefficient. Initial versions of the Draft stated that “*TARGET* parameter *MUST* be set to 25 milliseconds and *GAIN* *MUST* be set so that max ramp up rate is the same as for TCP.”. The selection of a constant and moreover specific value for *TARGET* is quite controversial, as it is clear that non-compliant implementation with a larger target delay are advantaged and could introduce severe fairness issues. Moreover, the default value for *TARGET* changed to 100 milliseconds in latter versions of the Draft, so to be compliant with the value specified in BEP-29 [71]. Concerning the second parameter, we set it to $\text{GAIN} = 1 / \text{TARGET}$, choice that we motivate in the next section.

We underline here a nice property of the PID controller: the window growth is directly proportional to the difference between the queuing delay estimate and the target off_target . In this way, when the queuing delay is close to the target, the controller response will be near zero, thus avoiding undesirable oscillations. Conversely, when the estimation is far from the target, the controller will increase the window faster and hopefully converge earlier.

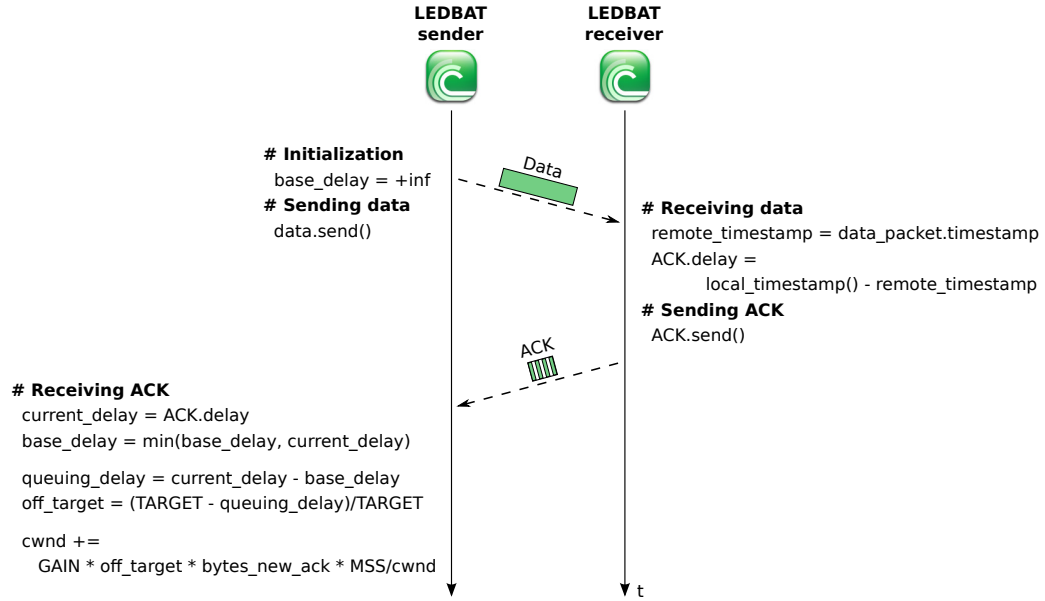


Figure 3.1: Pseudocode of the LEDBAT sender and receiver operations.

3.1.3 TCP Friendliness Consideration

An important goal of LEDBAT concerns its ability to yield to TCP traffic when sharing the same bottleneck resources. LEDBAT should be able both to detect the traffic already present on links, and to yield quickly to newly incoming connections.

At the same time, LEDBAT must avoid starvation. In fact if LEDBAT always yielded to any kind of traffic, even to the one generated by non interactive application (e.g., a long-lived FTP transfer), the performance degradation perceived by users may convince them to simply revert to TCP-based transfers, regardless of LEDBAT potential advantages.

A first necessary condition for TCP friendliness, is that LEDBAT *should never ramp-up faster than TCP*. Since LEDBAT increases its congestion window of the largest amount when the delay estimate is zero (notice also that estimated delay can never be negative), by selecting $GAIN=1/TARGET$ we guarantee that LEDBAT never ramps-up faster than TCP, as its maximum ramp-up speed is limited to one packet per RTT (i.e. like TCP in congestion avoidance).

A second requirement is that the delay-based LEDBAT congestion controller *should react earlier than loss-based TCP controller*: intuitively, if the former can ramp-down faster than loss-based connections ramp-up, it will yield to the latter. The Draft states that LEDBAT should “*yield at precisely the same rate as TCP is ramping-up when the queuing delay is double the target*”. Again our choice of $GAIN=1/TARGET$ fulfills this requirement: in fact, when the queuing delay is twice the target, LEDBAT will ramp-down at a rate equal to one packet per RTT, matching thus TCP congestion avoidance ramp-up speed.

A third final condition is that, *in case of loss, LEDBAT should behave like TCP does* (i.e., halve its congestion window). From all these considerations, one can derive that LEDBAT design follows a quite conservative approach, as in the worst case (when the queue estimation always equals zero) its most aggressive behavior simply degenerates into TCP.

3.1.4 Lower-than Best Effort transport protocols

In this section, we provide further details concerning two of the LBE protocols which are closer in spirit with LEDBAT, namely TCP-LP and NICE. To facilitate their comparison, we also report simple simulation results in Fig. 3.2, so to better visually highlight the relevant characteristics of each protocol. The top row of Fig. 3.2 reports the heterogeneous case where two flows employing different congestion control protocols are compared. The bottom row of Fig. 3.2 shows the time evolution of two flows employing the same LBE protocol assuming similar network conditions.

More precisely, for each $LBE \in \{TCP-LP, NICE, LEDBAT\}$ protocol, Fig. 3.2 depicts the temporal evolution of the $cwnd$ of different flows in two scenarios of a simple bottleneck topology. In the inter-protocol case (top row, labeled as TCP-LBE), low-priority protocols compete against a standard TCP flow, while in the intra-protocol case (bottom row, labeled as LBE-LBE) two LBE flows compete against each other. In the figure, link capacity is set to $C = 10$ Mbps, round-trip delay to $RTT = 50$ ms and the buffer is $B = 100$ MTU sized packets.

TCP-LP. TCP-LP measures one-way packet delays and employs a simple delay threshold-based method for early inference of congestion. More specifically, TCP-LP estimates the minimum D_{min} and maximum OWD D_{max} , filtering the instantaneous measure of the delay $D(t)$ by means of an exponentially weighted moving average $\tilde{D}(t)$ with smoothing parameter α , updated packet-by-packet. The smoothed average $\tilde{D}(t)$ and the condition for early congestion detection are:

$$\tilde{D}(t) = (1 - \alpha)\tilde{D}(t-1) + \alpha D(t) \quad (3.1)$$

$$\tilde{D}(t) > D_{min} + (D_{max} - D_{min})\delta \quad (3.2)$$

where $\delta \in (0, 1)$ is a custom threshold parameter. Throughout this study, we use the values $\alpha = 1/8$, $\delta = 0.15$ that are selected in [60] by means of simulation experiments.

In the absence of early-congestion indication, TCP-LP behaves like standard TCP Reno, i.e., performing an additive increase of the congestion window $cwnd$ as shown in Fig. 3.2-(b) and (f). Whenever an early congestion is detected, according to the rules outlined above, TCP-LP halves the congestion window and enters an inference phase by starting an inference timeout timer. During this period, TCP-LP only observes responses from the network and avoids increasing the congestion window. After this phase, if congestion persists it reduces the congestion window to zero and restarts the TCP Reno congestion avoidance scheme. Finally, in case of losses, TCP-LP behaves like TCP Reno.

NICE. NICE instead maintains a minimum round trip delay RTT_{min} and maximum RTT_{max} of the RTT delay. Congestion is detected when more than a given fraction ϕ of packets during the

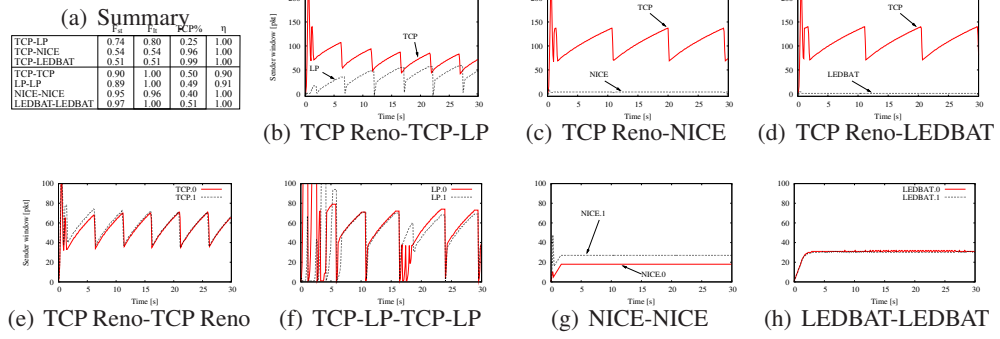


Figure 3.2: Low priority at a glance: Inter (top row) and Intra (bottom row) protocol interaction on a simple bottleneck.

same RTT experiences a delay exceeding:

$$RTT > RTT_{min} + (RTT_{max} - RTT_{min})\delta \quad (3.3)$$

where δ and ϕ are protocol parameters set to $\delta = 0.2$ and $\phi = 0.5$ as in [81]. Notice that (3.3) is the same formula of TCP-LP (3.1), but computed on the RTT variable, and using the fraction-trick instead of a moving average.

In the absence of congestion, NICE behaves like Vegas [27], whose congestion window dynamics are delay-based (and thus rather different from loss-based dynamics). Whenever early-congestion is signaled, NICE simply halves its congestion windows and sending rate, practically reintroducing the multiplicative decrease behavior. Finally, when a loss is detected it instead reacts like TCP Reno.

The fact that NICE inherits its congestion control behavior from Vegas [27], rather than from TCP Reno, has profound impact on the *cwnd* evolution: as observed in Fig. 3.2-(c) and (g), NICE shows a much smoother behavior as its throughput stabilizes around the effective link capacity. We point out that NICE allows *cwnd* to be a fraction of 1 by sending one packet after waiting for the appropriate number of RTTs: the use of fractional values for *cwnd* guarantees non-intrusiveness even in the case of many NICE flows sharing the same bottleneck.

LEDBAT. Finally, LEDBAT maintains a minimum OWD estimation D_{min} , which is used as base delay to infer the amount of delay due to queuing. LEDBAT flows have a target queuing delay τ , i.e., they aim at introducing a small, fixed, amount of delay in the queue of the bottleneck buffer. Flows monitor variations of the queuing delay $D(t) - D_{min}$ to evaluate the distance $\Delta(t)$ from the target as in (3.4):

$$\Delta(t) = \tau - (D(t) - D_{min}) \quad (3.4)$$

$$cwnd(t+1) = cwnd(t) + \gamma \Delta(t) / cwnd(t) \quad (3.5)$$

where τ, γ are protocols parameters that we study later on.

In the absence of early-congestion indication, i.e., when the target τ has not been reached yet, $\Delta(t) > 0$ in (3.4) and thus $cwnd$ grows as defined by (3.5). Notice that when the target is reached, $\Delta(t) = 0$ thus $cwnd$ settles.

Values of $\Delta(t) < 0$ are perceived as early-congestion indication (i.e., other traffic is increasing the queuing delay $D(t) - D_{min}$), to which LEDBAT reacts by reducing $cwnd$ proportionally to the offset from the target according to (3.5). Finally, in case of losses, it behaves like TCP Reno.

Overall, LEDBAT shares similarities with, and exhibit differences from, the other LBE protocols: (i) as TCP-LP, it relies on One-Way Delay estimation to detect congestion, but unlike TCP-LP it does not employ a smoothing average; (ii) as NICE, its congestion controller is based on the delay, but unlike NICE it employs a PID controller in order to reach (or deviate from) the target delay. As we can see from Fig. 3.2, the behavior of LEDBAT is however closer to NICE than to TCP-LP.

3.2 LEDBAT performance

In this section, we report results gathered with our open-source implementation (available at [7]) of the LEDBAT controller in the Network Simulator ns2 [11]. We start by illustrating some telling examples of the LEDBAT dynamics in simple cases, incrementally adding complexity to refine the picture later on. To carry out the comparison, we had to implement also NICE, while TCP Reno and TCP-LP protocols are already implemented in the simulator.

The code for LEDBAT is build on top of the `tcp-linux` module, so it can be used either by the simulator or loaded directly into the Linux kernel and exploited by real application as a novel protocol at transport layer.

3.2.1 Reference scenario

As reference scenario, we consider a bottleneck link of capacity C Mbps and buffer size B packets. For the sake of simplicity, we assume that all transceivers adopt $P = 1500$ Bytes fixed-size packets. The traffic is backlogged and flows in a single direction, while acks are not delayed, dropped nor affected by cross-traffic on their return path. All flows have the same round trip time $RTT = 50$ ms, half of which is due to the propagation and transmission delay components of the bottleneck link (i.e., a one-way base delay of 25 ms).

In our analysis we restrict our attention to a simple high-speed access scenarios, with a link of $C = 10$ Mbps capacity for downlink/uplink, and different buffer sizes $B \in [10, 100] \subset \mathbb{N}$ packets. Notice that, once fixed the link capacity C and the packet size P , we can express the queuing delay TARGET in terms of either a time-lapse or bytes (and packets). Denoting for short the TARGET as τ , in the following we will refer indifferently to the queuing delay in terms of time-lapse $\tau_T = 25$ ms or packets $\tau_P = \tau_T C / 8P$ (with capacity expressed in kbps and packet size in bytes). For instance in our high-speed scenario, $\tau_T = 25$ ms corresponds to $\tau_P = 20.8$ packets. Thus, a buffer size of $B = 40$ packets, almost equal to the bandwidth-delay product, can accommodate twice as much queuing delay than the LEDBAT TARGET τ .

Table 3.1: Fairness, breakdown and efficiency for two flows in homogeneous conditions.

	F_{st}	F_{lt}	TCP%	η
TCP - LP	0.74	0.80	0.25	1.00
TCP - NICE	0.54	0.54	0.96	1.00
TCP - LEDBAT	0.51	0.51	0.99	1.00
TCP - TCP	0.90	1.00	0.50	0.90
LP - LP	0.89	1.00	0.49	0.91
NICE - NICE	0.95	0.96	0.40	1.00
LEDBAT - LEDBAT	0.97	1.00	0.51	1.00

The initial tests consider only one TCP and one LEDBAT flow, which start simultaneously and share the network resources for 120 seconds. We then perform a sensitivity analysis of LEDBAT, to assess the impact of parameters τ and γ on the system performance. We carry out this analysis in both (i) an inter-protocol case, where a TCP Reno flow and a LEDBAT flow share the bottleneck and (ii) an intra-protocol case, where a two LEDBAT flows compete against each other. The aim of (i) is to determine whether τ and γ offer the chance to tune the level of LBE priority in LEDBAT, while (ii) aims at verifying whether unfairness may arise among legacy LEDBAT implementations (e.g., different releases of the same code, different implementations or parameter settings, etc.).

We then focus on a comparison of TCP and LBE protocols, again considering two cases: (iii) a single TCP flow shares the bottleneck with a varying number of homogeneous LBE flows (i.e., same LBE protocol) and (iv) several heterogeneous LBE flows compete against each other. In both cases, our aim is to evaluate the level of low priority of LBE protocols. Finally, we consider more realistic scenarios in (v) by taking into account the impact of RTT heterogeneity on LBE performance.

3.2.2 Evaluation metrics

Performance evaluation is carried out considering different metrics, that relate to either network-centric (e.g., efficiency, average queue size) or user-centric performance (e.g., fairness, packet loss rate). To illustrate this, Tab. 3.1 summarizes the performance of flows in corresponding scenarios in terms of some of these metrics.

Bottleneck link efficiency (η) is the primary network-centric metric, and expresses the link utilization as the ratio between the sum of the throughput values x_i achieved by all flows over the available capacity:

$$\eta = \frac{\sum_i x_i}{C} \quad (3.6)$$

Average queue occupancy index (B) is computed averaging buffer occupancy during the simulation (measured at each enqueue event in the buffer), and normalizing the value over the buffer

size for convenience:

$$B = \frac{E[B]}{B_{max}} \quad (3.7)$$

Whenever the buffer overruns and packets are dropped, all protocols drastically reduce their sending window: *packet loss probability* (P_l) therefore relates to user-performance, and is computed as the ratio of the dropped packets over the total number of packets sent on the link.

We further express the system performance using two metrics apt at describing how the link resources are shared among flows. To gauge the impact of LBE on TCP, we define *TCP breakdown* ($TCP_{\%}$) as the TCP Reno traffic share percentage over the total amount of data exchanged on the link:

$$TCP_{\%} = \frac{\sum_{j \in TCP} x_j}{\sum_i x_i} \quad (3.8)$$

Finally we further describe the capacity share in terms of *Jain's fairness index* (F), defined as:

$$F = \frac{(\sum_{i=1}^N x_i)^2}{N \cdot \sum_{i=1}^N x_i^2} \quad (3.9)$$

where N is the number of considered flows and x_i is the rate of the i -th flow. In the best case, when all flows get a fair share of the resources, F is equal to one, while in the worst one, namely when a single flow exploits all the link, it is equal to $1/N$ (for instance when considering a single LBE flow competing against TCP).

We compute the fairness index over both the whole flow duration and over a smaller time scales (considering a temporal window of 20 RTT, or equivalently 1 s): we refer to *long-term fairness* (F_{lt}) in the first case, and to *short-term fairness* (F_{st}) in the latter one. Notice that the ability to achieve short-term (vs long-term) fairness may have rather different implications, e.g., if we consider the case of several P2P flows measuring throughput to perform peer selection (as long-term fairness may not be sufficient and significantly biases peer decisions).

3.2.3 Homogeneous Initial Conditions

Our investigation starts by considering a LEDBAT flow competing for the same bottleneck resources with either i) a TCP or ii) another LEDBAT flow. For the time being, we disable slow-start in both implementation as we are interested in the interaction of the LEDBAT PID the TCP AIMD controllers. We let both flows start at $t = 0$, when the queue is empty and no other traffic is present on the link, so that LEDBAT is able to accurately measure the base delay.

Fig. 3.3-(a) shows the temporal evolution of the LEDBAT and TCP windows (top) as well as of the queue length (bottom), with a buffer size of $B = 40$ packets. We recognize the usual TCP sawtooth behavior, which defines a number of cycles. During the initial ramp-up ($t < 2$ s), LEDBAT and TCP windows grow *nearly* at the same speed of one packet per RTT. LEDBAT grows at its maximum speed because the available link capacity keeps the queue empty. As soon as queue builds up, the LEDBAT linear controller reacts accordingly by slowing down the increase of its

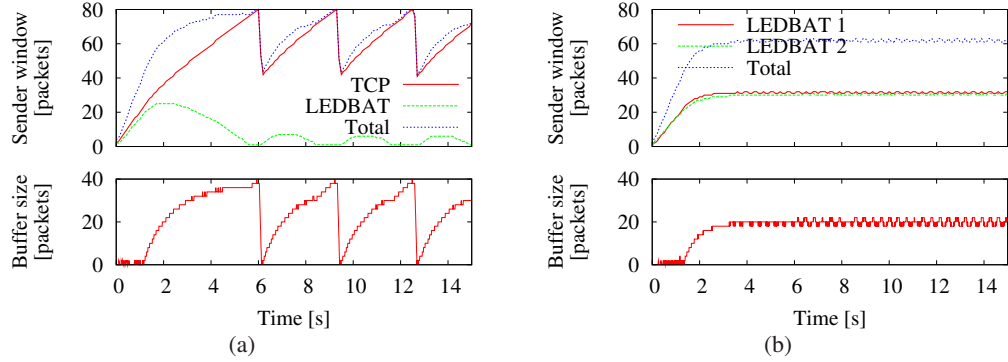


Figure 3.3: Temporal evolution of the sender window (top) and of the queue size (bottom) for (a) TCP-LEDBAT and (b) LEDBAT-LEDBAT interaction.

sending rate, while TCP behavior remains instead unaltered. Soon after $t = 2$ s, LEDBAT hits the $\tau_P = 20.8$ packet target, and halts the window growth, so presenting a flat sender window curve. TCP, instead, continues its additive increase, so that the queue keeps building up until the queuing delay exceeds the target: the LEDBAT controller, unlike TCP, reacts by decreasing its sending rate, finally reaching the minimum rate of one packet per RTT just before $t = 6$ s.

Slightly afterwards, TCP causes a buffer overflow: consequently, TCP abruptly decreases its sending rate by halving its congestion window. The capacity drains the queue empty, giving thus start to a new cycle. In fact, LEDBAT detects the delay reduction and reacts by opening its window again. However, in this cycle TCP has an initial window size of about 40 packets, which means that it can create queuing sooner with respect to the previous cycle. Therefore, LEDBAT window growth is slower, the TARGET delay is hit earlier (at about $t = 7$ s) and also the window shrink phase appears much shorter. When TCP is again the sole sender on the link, it increases its sending rate until a new loss happens, which in turn triggers the start of a new cycle.

Fig. 3.3-(a) confirms that, as LEDBAT reacts to congestion *earlier* than TCP by estimating the queuing delay, it is able to yield to TCP, which can *work undisturbed*. In fact, losses are due to the normal AIMD dynamic of TCP rather than to the LEDBAT-TCP interaction. Fairness in this case equals $F = 0.65$, with TCP transferring 6 times as much data with respect to LEDBAT during the same timeframe. Fig. 3.3-(a) also reports the sum of both TCP and LEDBAT sender windows, which represents an estimate of the instantaneous link utilization. When TCP and LEDBAT coexist on the link, its *utilization increases* with respect to the case where TCP is alone – in the figure utilization increases by 16%, compared to the case where TCP is alone on the bottleneck.

Fig. 3.3-(b) shows a similar experiment, in which two LEDBAT sources start competing at $t = 0$ for the bottleneck resources. In this case, both senders employ a linear controller and are able to share resources fairly ($F > 0.99$) and efficiently (efficiency is only 0.7% less than in the Fig. 3.3-(a) case). As expected, once the delay target is reached, LEDBAT sources settle (since the

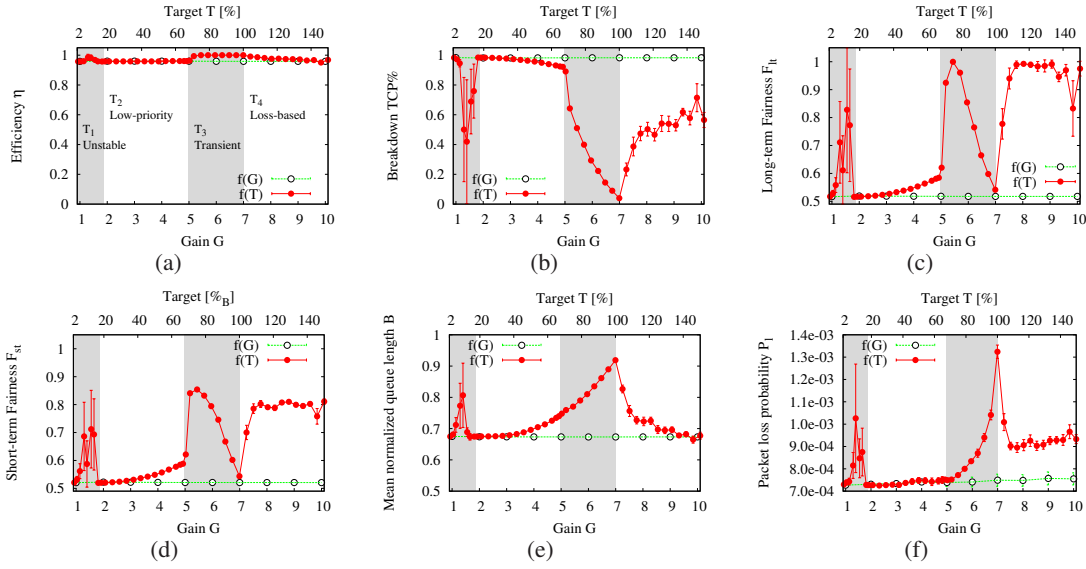


Figure 3.4: LEDBAT vs TCP Reno: Inter-protocol sensitivity analysis, for varying LEDBAT target T and gain G parameters.

offset from the target is zero, and so the controller response). Notice also that, since the two sources started together, they measured the same base delay at $t = 0$. Therefore, each sender independently settles when measuring a queuing delay equal to the target, thus it is actually responsible only for half of buffer occupancy.

3.2.4 LEDBAT Sensitivity Analysis

3.2.4.1 Inter-protocol: LEDBAT vs TCP

We start our sensitivity analysis by considering two flows, a standard TCP Reno and a LEDBAT one, that start simultaneously and vary the values of parameters τ and γ one at a time. Notice that the standardization Draft *does not specify any value* for the gain parameter γ . Conversely, the Draft *specifies a mandatory value* for the TARGET parameter equal to $\tau = 25$ ms. This choice of τ is somewhat arbitrary, and based on experimental observations (whose results are however unreported so far) or motivated by practical constraints (e.g., today's limits in the precision of the delay measurement, etc.), rather than being based on concrete foundations. As such, τ is often referred to as “magic number” with a deprecatory sense in LEDBAT WG discussion [6]: therefore, we believe that a thorough exploration of the impact of the above parameters is necessary, which we carry out by simulation. Moreover, later versions of the Draft specify a TARGET equal to 100 ms which can further increase the confusion in case of early and late implementations of the algorithm.

For convenience, we re-express the target delay parameter τ in terms of buffer percentage as

$T = \tau C / (SB_{max})$, and explore the range $T \in [2, 150]\%$, corresponding to $\tau \in [2.4, 180]$ ms. For reference purposes, notice that the mandatory Draft value $\tau = 25$ ms correspond to $T = 20\%$, while a full buffer occupancy $T = 100\%$ is attained when $\tau = 120$ ms.

Fig. 3.4 reports the simulation results for each of the metric $f \in \{\eta, TCP\%, F_{st}, F_{lt}, B, P_l\}$ described early in Sec. 3.2.2 arranged as one per plot. In each plot, we report two curves, namely $f(G)$ and $f(T)$. The $f(G)$ curve reports how $f(\cdot)$ varies as a function of the gain $G \in [1, 10]$ (on the bottom x-axis), when target is fixed to $\tau = 25$ ms. The $f(T)$ curve instead reports how $f(\cdot)$ varies as a function of the target $T \in [2, 150]\%$ (on the top x-axis), when gain is fixed to $G = 1$.

From all the subplots we can see that, for all metrics, the $f(G)$ curve is roughly flat, i.e., the gain parameter only minimally affects the behavior of the LEDBAT protocol in this case. This can be explained by the fact that, as LEDBAT is designed to yield to TCP, it will yield irrespectively of G . The gain value thus only affects the speed at which LEDBAT will yield, which quickly happens for any value of G .

Therefore, from now on we restrict our attention to the impact of the target parameter, and analyze the behavior of the $f(T)$ curves. In Fig. 3.4-(a) we can see that the efficiency η is only slightly influenced by the variation of the target and remains always close to the total link capacity. This is expected, as even if the target is misconfigured, either LEDBAT or TCP Reno can take advantage of the unused bandwidth, which result in an overall efficient use of the link capacity.

Considering instead the $TCP\%$ reported in Fig. 3.4-(b), we can identify four working regions. When the target is very small $T_1 \in [2, 20]\%$ the LEDBAT protocol is not always able to reach the target delay, which leads to shaky $TCP\%$ behavior. In a second region $T_2 \in [20, 65]\%$, LEDBAT completely yields to the TCP Reno flows, working in low-priority mode and thus attaining its goal. In a third region $T_3 \in [65, 100]\%$, LEDBAT aggressively starts to erode bandwidth to the TCP Reno flow: this causes losses in the TCP Reno flow, which progressively backs off; as a consequence, the $TCP\%$ starts decreasing until LEDBAT has the monopoly of the buffer $T = 100\%$ and TCP Reno starves ($TCP\% \simeq 0\%$). Finally, in the fourth region $T_4 > 100\%$ the target exceeds the buffer size: in this case, LEDBAT falls back in the TCP Reno-like loss-based behavior, increasing the sending rate until a loss occurs, which immediately drop down its rate. As a consequence, the breakdown is now more similar ($TCP\% \simeq 50\%$).

Similar considerations can be gathered by looking at the long-term F_{lt} or short-term F_{st} fairness plots shown in Fig. 3.4-(c) and Fig. 3.4-(d) respectively: indeed, an even breakdown corresponds to maximum fairness ($F_{st} \simeq 1$) while to an uneven breakdown, favoring either TCP Reno ($TCP\% \simeq 100\%$) or LEDBAT ($TCP\% \simeq 0\%$), always corresponds to minimum fairness values ($F_{st} \simeq 1/2$).

From Fig. 3.4-(e) and Fig. 3.4-(f) we see that, as expected, increasing the target the average buffer occupancy rises, as the increased traffic due to LEDBAT sums up with the TCP Reno one. Losses increase as well reaching a peak for $T = 100\%$, corresponding to LEDBAT maximum aggressiveness; afterward LEDBAT is in loss-mode, and the scenario degenerates into two TCP Reno flows sharing a bottleneck.

Overall, the sensitivity analysis suggests that, although LEDBAT spans a wide range of low-priority levels (especially in the third region), its tuning is highly impractical. Indeed, the support of

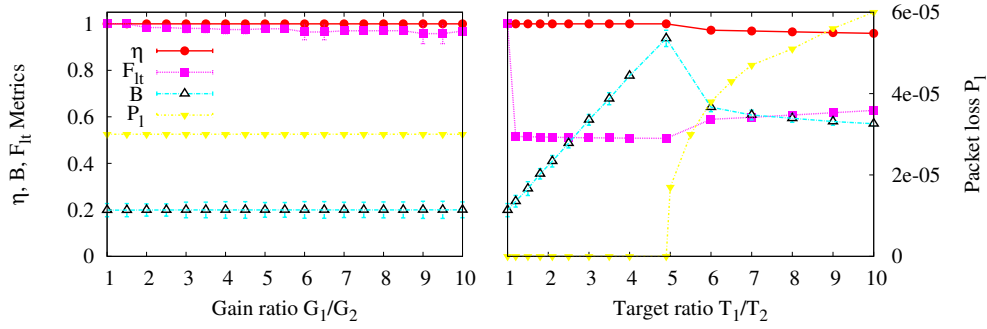


Figure 3.5: LEDBAT vs LEDBAT: Intra-protocol sensitivity analysis, for varying LEDBAT target T_1/T_2 and gain G_1/G_2 ratios.

target values $T_3 \in [65, 100]\%$ is very small, meaning that a small variation of T lead to completely different scenarios, where either LEDBAT or TCP Reno exhibit starvation. Moreover, the actual values of τ yielding to a specific level of low-priority depends on network parameter (e.g., capacity C , buffer size B) and are likely affected from other factors as well (e.g., number of TCP Reno flows, heterogeneous RTT, etc.)

3.2.4.2 Intra-protocol: LEDBAT vs LEDBAT

We pursue our sensitivity analysis by considering two LEDBAT flows with heterogeneous settings sharing the same bottleneck link. We perform two sets of experiments, varying either (i) the gain ratio G_1/G_2 of the two flows when $G_2 = 1, \tau = 25$, or (ii) the target ratio T_1/T_2 when $T_2 = 20\%, \gamma = 1/\tau$. In both cases, the ratio varies in the $[1, 10]$ range. Results of the sensitivity analysis are reported in Fig. 3.5, which depicts the packet loss rate P_l (right y-axis), the average buffer size B , the efficiency η and the fairness F_{lt} (left y-axis) as a function of the G_1/G_2 gain ratio (left plot) and T_1/T_2 target ratio (right plot).

As in the previous case, the impact of the gain is very modest, even in the case of a 10-fold factor. This phenomenon has an intuitive explanation. Consider indeed, that a flow with the largest gain will start moving faster than the other flow toward the target. However, after the first flow increases its window, the convergence speed toward target will slow down, since the differences between the target and the measured delay is now smaller for the first flow than for the second. In other words, the difference in the delay offset in (3.5) compensates for differences in the gain factor γ .

Conversely, even slight differences in the target settings may have strong consequences as it can be seen in the right plot of Fig. 3.5. Indeed, as soon as $T_1/T_2 > 1$ it can be seen that the fairness immediately drops to its minimum value $F_{lt} = 0.5$. This is due to the fact that flows with higher-target are always more greedy than their lower-target counterpart. As a matter of fact, if

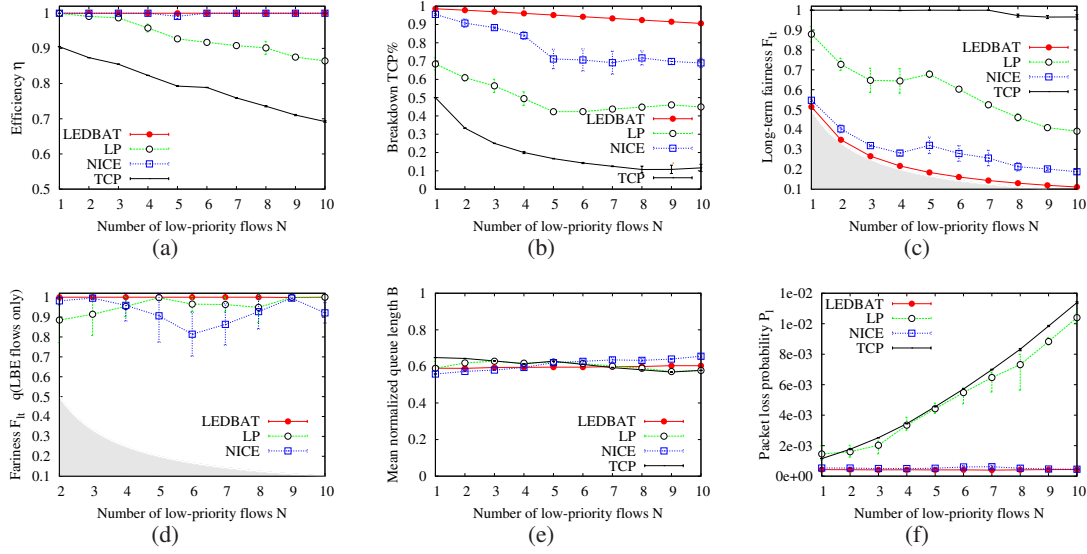


Figure 3.6: LBE against one TCP flow: Impact of the number LBE flows on the system performance.

both flows start at the same time, they both measure the same base delay, and the higher-target flow converges faster to its target and stabilizes: as the amount of queuing is now larger than that of the less aggressive flow, the latter backs off and starves. This holds until $T_1 + T_2 > 100\%$ which happens when $T_1/T_2 = 5$ given that $T_2 = 20\%$, in which case both LEDBAT flows may experience packet drops: nevertheless, higher-target flow will always be advantaged prior than losses occur, and so the unfairness persists.

Overall, we see that gain and target parameters have rather different effects: on the one hand, provided that LEDBAT flows have the same target, differences in gain do not entail any unfairness among flows. On the other hand, even a small difference in targets produces an extremely unfair situation: this is a delicate point, which we believe deserves further attention in the future.

3.2.5 LBE protocols comparison

3.2.5.1 LBE against TCP

We now fix LEDBAT parameters and consider a larger set of LBE protocols in the comparison. Following our sensitivity analysis, we know that the selection of γ , rather than τ , is less critical: we set then $\gamma = 1/\tau$, and use the mandatory value $\tau = 25$ ms which we verified to be a robust choice.

We consider a typical scenario where N , ($N \in [1, 10]$) low-priority flows (e.g., due to P2P or other services) share the same bottleneck with a single TCP Reno connection, (representative of a

generic high-priority service), for a total of $N + 1$ flows. We perform several sets of simulations separately, considering each time a different LBE protocol. For reference purpose, we also simulate the case where $N + 1$ TCP Reno flows share the same bottleneck.

Results for the common set of metrics are reported in Fig. 3.6. Considering efficiency η in Fig. 3.6-(a), we see that delay-based NICE and LEDBAT are able to fully utilize the spare bandwidth left by TCP Reno. Conversely, in the TCP-LP or TCP Reno cases, losses entail a reduction in efficiency.

Breakdown $TCP\%$ reported in Fig. 3.6-(b), states that e.g., in the $N = 10$ LEDBAT case, TCP Reno consumes about 90% of the link capacity (since $\eta \simeq 1$), leaving thus the $N = 10$ LEDBAT flows a mere 1% of the capacity each. Comparing this result with NICE (about 3% each) or TCP-LP (about 5% each) under the same $N = 10$ settings, we gather that LEDBAT achieves the lowest priority, closely followed by NICE. This is further exacerbated from the long-term fairness plot of Fig. 3.6-(c), showing that in the LEDBAT and NICE cases fairness approaches the minimum possible value (i.e., the shaded region indicates values that fairness cannot achieve since they are smaller than the lower bound $\frac{1}{N+1}$).

The plot in Fig. 3.6-(d) depicts instead the long-term fairness F_{lt} evaluated over the N LBE flows only. It can be seen that fairness is always high, meaning that generally the excess that remains after the TCP breakdown of Fig. 3.6-(b), is evenly shared among LBE flows. Notice that fairness among LBE flows is however lower in the case of NICE, where apparently some LBE flow opportunistically take advantage of the others.

Finally, average occupancy index B and packet loss P_l are reported in Fig. 3.6-(e) and (f) respectively. Again, delay-based versus loss-based congestion control principles are remarkably different, which is especially true in the case of the loss curve: interestingly, despite its low priority aim, the amount of loss induced by TCP-LP is strikingly similar to that of classic TCP Reno. Delay-based versus loss-based difference, although less evident, also reflects on the queue size: indeed, TCP Reno and TCP-LP average queue size decrease when number of flows and losses increase; conversely, queue occupancy in the NICE case slowly rises for increasing N , and is practically unaffected by N in the LEDBAT case.

3.2.6 LBE against LBE

In order to investigate the mutual interaction of the different lower-priority protocols, we define a heterogeneous scenario in which several LEDBAT, TCP-LP and NICE flows contend the same bottleneck link. We perform different tests in which an increasing number of flows is considered, from 1 to 5 for each flavor (which corresponds to a total of 3 to 15 flows). As reference, we perform also the corresponding experiment with the same number of TCP Reno flows only (i.e., 3 to 15 TCP Reno flows). We choose for all the LEDBAT flows, the standard parameters values, namely $\tau = 25\text{ ms}$ and $\gamma = 1$.

Let us start by examining the efficiency η and average normalized buffer length B , which are reported in Fig. 3.7-(a). Looking at the efficiency, we can see that in the heterogeneous LBE scenario, flows are able to utilize the available resource fully, with η always close to its maximum.

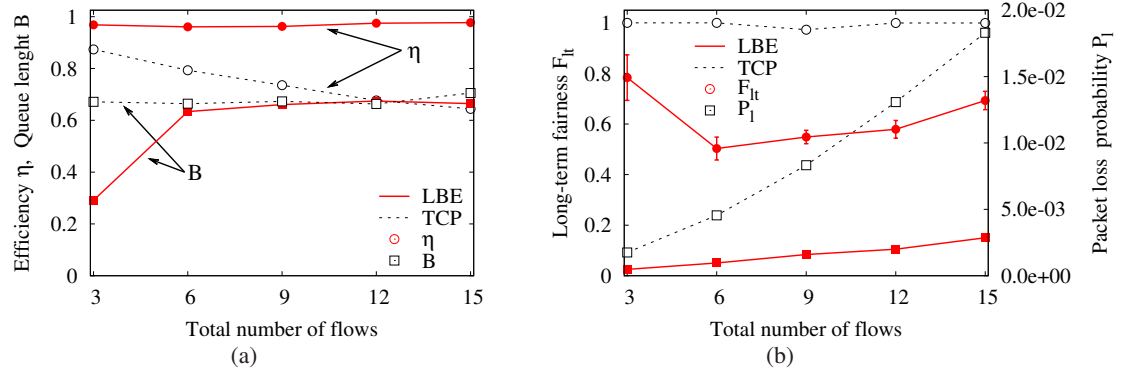


Figure 3.7: LBE against LBE: TCP-LP, LEDBAT and NICE competing at the same bottleneck, compared with the same number of TCP Reno only flows.

On the contrary, the efficiency in the case of all TCP Reno flows progressively decreases as long as the number of competing flow increases, due to the typical synchronization behavior of the protocol after loss. Looking at the normalized average queue size we can notice that the average $B \simeq 2/3$ is not affected by the number of flows in the TCP Reno case. In the LBE case instead, average queue size approaches that of TCP Reno only when at least two flows per protocol insist on the bottleneck. When only a total of three LBE flows are competing for the resource, a rather unexpected phenomenon arises: in this case, LEDBAT often forces TCP-LP in low-priority mode and is thus able to exploit a significant part of the resource. As a consequence, the average queue size B reflects the LEDBAT target τ , plus the contributions due to TCP-LP and NICE. When more than two TCP-LP flows are instead present on the bottleneck, their behavior synchronizes and is perceived as more aggressive by LEDBAT: in this case, it is rarer that *both* TCP-LP flows enter into inference mode at exactly the same time, thus LEDBAT has fewer opportunities to profit from the resource.

Packet loss probability P_l and long-term fairness F_{lt} are reported in Fig. 3.7-(b). Concerning packet loss, since $2/3$ of the total flow number consists of delay-based protocols, the loss rate is clearly lower than the TCP Reno reference case. Long-term fairness performance shown in Fig. 3.7-(b) is instead better understood by considering also the throughput breakdown reported in Fig. 3.8, in which each bar represents the percentage of traffic due to a particular LBE protocol. As expected, fairness between heterogeneous LBE flows is lower than that of homogeneous TCP Reno connections, but is however higher than the LBE-TCP Reno performance earlier reported in Fig. 3.6-(c). In particular, maximum LBE fairness is achieved when only one flow per each LBE flavor is considered: from Fig. 3.8 we see that TCP-LP and LEDBAT performance are very close in this case, which raises the fairness metric. When the number of low-priority flows increases, the fairness instead decreases due to a higher aggressiveness of the TCP-LP protocol.

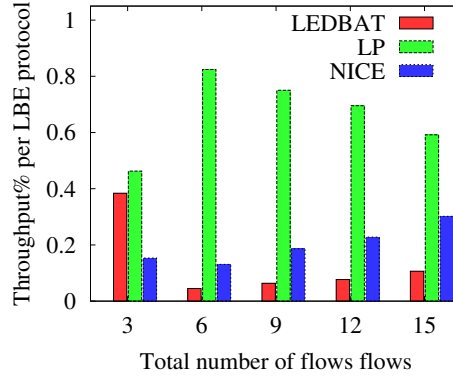


Figure 3.8: Traffic share with an equal number of TCP-LP, LEDBAT and NICE flows competing for the same link.

3.2.7 Impact of RTT heterogeneity

Finally, we report on the impact of RTT heterogeneity in Fig. 3.9. We consider two flows, of the same protocol type (LBE or TCP) sharing the same bottleneck, that have a different round trip delay expressed by the RTT_1/RTT_2 ratio. We perform simulations separately for each protocol, exploring the $RTT_1/RTT_2 \in [1, 10]$ range; RTT_1 is increased by adding propagation delay to the return path, so that OWD estimation on the forward path is not affected. The left plot of Fig. 3.9 reports the long term fairness F_{lt} , while right plot reports the efficiency η as a function of the RTT ratio.

An interesting remark to make concerning the fairness metric is that only NICE, by virtue of its inheritance of Vegas [27] congestion control, provides fairness in the case of heterogeneous RTT settings. However, this comes at the price of a reduced efficiency, since in order to be fair, the more aggressive small-RTT flow has to slow down its rate to match that of the large-RTT flow. Efficiency loss happens also in the case of TCP-LP and TCP Reno, despite their inability to offer fairness. Finally, LEDBAT realized a perfectly efficient system, which comes at the price of a totally unfair share of the resources. In fact the small-RTT flow is able to reach its target first, due to the faster feedback, whereas the second flow will see a queuing delay (due to the small-RTT flow) equals to its target, and will thus settle in a starvation state.

3.3 Summary

In this chapter, we reported on an initial evaluation of LEDBAT, which is designed for low-priority data transport and aims at being friendly and non-intrusive toward other protocols (such as TCP, VoIP and gaming), while being able to effectively exploit the available resources at the same time. Results gathered in the homogeneous initial conditions show that LEDBAT does not interfere with

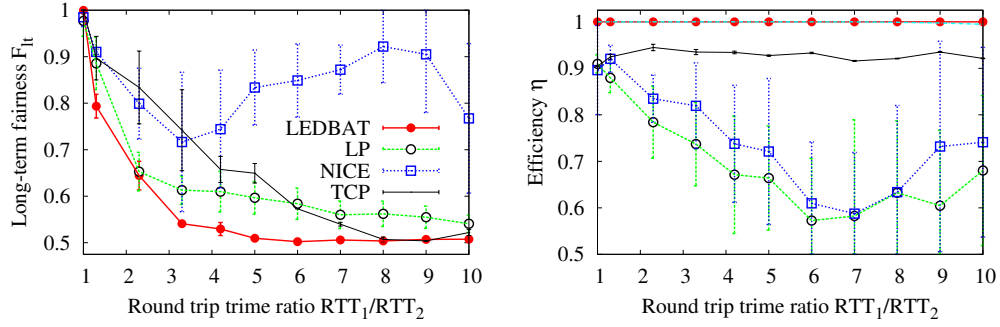


Figure 3.9: Impact of the RTT heterogeneity.

the transmission of data using TCP, but it allows an increase of the network utilization. Moreover the new protocol exhibits an optimal intra-protocol fairness, in the case two flows start at the same time and compete one against the other for the bottleneck.

Then, we analyzed the behavior of different LBE transport protocols. By means of simulation, we carried out a thorough comparison of LEDBAT, TCP-LP and NICE, studying the impact they have on TCP Reno traffic, as well as their mutual impact. The sensitivity analysis of LEDBAT reveals the difficulty in tuning its behavior, and especially its level of priority with respect to TCP Reno by means of a simple adjustment of its gain γ and target τ parameters. Indeed, the gain has practically no influence, while the impact of target can not be controlled, as changes in the system performance are too steep. Also, we see that gain and target parameters have rather different effects if we consider the coexistence of legacy LEDBAT flows with heterogeneous settings: on the one hand, provided that LEDBAT flows have the same target, differences in gain do not entail any unfairness among flows; on the other hand, even a small difference in targets results in an extremely unfair situation. We conclude that tuning LEDBAT is thus a delicate point, which deserves further attention in the future, which holds true even when heterogeneous network settings are considered. For this reason, we carried out a sensitivity analysis on the target parameter also from the BitTorrent swarm point of view, as presented in Chap. 6.

Our comparison study shows that LEDBAT achieves the lowest possible priority with respect to NICE and TCP-LP. Moreover, we find that TCP-LP inherits from its loss-based design a higher aggressiveness than the delay-based NICE whose degree of low-priority sits thus in between LEDBAT and TCP-LP. Interestingly, we point out that there are also limit cases (e.g., only an TCP-LP, LEDBAT and NICE flows sharing the same bottleneck) in which the low-priority degree can exhibit unexpected behavior (i.e., as LEDBAT is in this case as aggressive as TCP-LP).

Overall, the LEDBAT protocol seems to keep its promises and successfully meets some of its design goals. However in the next chapter we review the algorithm under heterogeneous initial conditions, which highlight some potential fairness issue, that need to be dealt with.

Chapter 4

Rethinking LEDBAT for fairness

In this chapter we focus on the latecomer advantage issue that arise in multiple flow scenario, when two LEDBAT flows start at different time; in such condition, the newly arriving connections can starve the already existing flows and take over all the bottleneck capacity. The fairness issue has been subject of an extended literature, as we report in Sec. 4.1. The same unfairness issue is highlighted at first via `ns2` simulation, with an approach similar to the one presented in Chap. 3, and then using a real world implementation of LEDBAT in Sec. 4.2. Then we investigate the design properties of LEDBAT using an analytical model and highlight the root cause of the problem, that lies in the linear controller adopted by the protocol. For this reason, we propose in Sec. 4.3 modifications to the congestion window update mechanism, introducing *fLEDBAT* for *fair LEDBAT*, that aim at solving this issue, thus guaranteeing intra-protocol fairness and efficiency. Closed-form expressions for the stationary throughput and queue occupancy are provided via a fluid model, whose accuracy is confirmed by means of packet level simulations in Sec. 4.4. In Sec. 4.5 we study the impact of different traffic models (e.g. chunk-by-chunk and backlogged data transfer) on *fLEDBAT*, and in Sec. 4.6 we perform a sensitivity analysis varying the protocol parameters. Then we compare in Sec. 4.7 the performance of *fLEDBAT* and LEDBAT in a P2P-like scenario, that mimics a chunk transfer between peers in a BitTorrent swarm, considering heterogeneous network conditions and background traffic. Finally, we summarize the content of the chapter in Sec. 4.8.

Differently from [28], the solution proposed in this chapter is more mature and it exploits for its validation not only simulations, but also a fluid approach, analytical techniques and experimental measurement. Moreover, the simple solutions to the fairness issue proposed in [28] partly failed in meeting the efficiency goal, which we instead successfully address with *fLEDBAT*.

4.1 Related work

Related work has already tackled the intra-protocol fairness issue affecting delay-based congestion control algorithms. In particular, regarding Vegas [27] which is the first example of such a family of techniques, the problem was pointed out [67] in relation to (i) to route changes and (ii) to persistent

congestion when multiple concurrent Vegas flows compete at the same bottleneck. The latter is the same malfunctioning we spotted in the original LEDBAT design: latecomer flows overestimate the base delay because of the queuing delay old flows have already put in the buffer. LEDBAT proposers clearly took advantage of this literature when designing the protocol, for instance when they adopted the same solution of [67] to the rerouting problem (i.e., by using only the recent history of delay observations for the base delay estimation), but they neglected the latecomer advantage.

To solve the fairness issue, researchers have followed various approaches: on the one hand, some works try to improve the estimation of the base RTT [38, 46, 84]; on the other hand, others propose techniques to achieve fairness in spite of the base delay estimation error [26, 48, 61, 62]. The first type of work usually relies on additional support from the network to correct the measurement: for instance, [84] uses an out-of-band priority packet which skips the queue and provides a good estimation of the base delay; [46], instead, adapts its parameters according to the number of congested routers on the path, thus relying on their feedback. Authors of [26, 48] follow the opposite approach and prove that a particular choice of parameters allows flows to converge to a fair share of available bandwidth. The delay based algorithm proposed in [62] was also shown capable of dealing with noisy delay measurement, thanks to the careful choice of the controller function. Finally, [61] proposes a new delay based AIMD algorithm and choose a backoff factor which avoids measurement errors. Our work is the first to study this issue for a LBE protocol and to achieve together efficiency, fairness and lower-priority by reintroducing the multiplicative decrease component, for we correctly identify the root cause of unfairness in the additive decrease component [56] rather than in the measurement error.

4.2 Motivation

To highlight the relevance of this study, we show in this section the conditions in which the unfairness situation arises, by means of both simulation and traffic measurement in a controlled testbed.

4.2.1 Latecomer in simulated environment

The simulation test are performed considering different start times for different sources. This implies that each sender will measure a different base delay at startup, gathering also a different estimate of the queuing delay. Indeed, assume that the first flow starts at time $t_1 = 0$, while the second starts at time $t_2 = t_1 + \Delta T$. In case the queuing delay at t_2 is not zero but equal to $t_Q(t_2)$, the second source will over-estimate the base delay $t_B(t_2)$ with respect to the one measured by the first source as $t_B(t_2) = t_B(t_1) + t_Q(t_2)$. So, the second source will set its target to a value higher than the first one, increasing the chances of a buffer overflow.

In case of interaction between LEDBAT and TCP, heterogeneity of initial conditions has a negligible impact. To convince of this, consider that, whenever LEDBAT starts first, it will be able to correctly estimate the base delay, and then to yield to TCP. Conversely if the LEDBAT flows starts later at t_2 , it will over-estimate the base delay by the amount of TCP packets present in the buffer. This will in turn make LEDBAT under-estimate the queuing delay, resulting in an increased

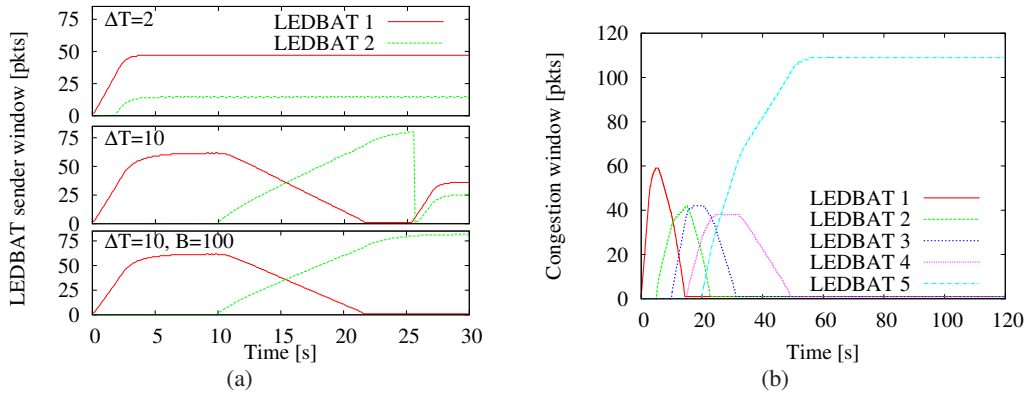


Figure 4.1: LEDBAT vs LEDBAT: Time evolution of congestion window for different initial condition and late-comer advantage phenomenon with (a) two flows and (b) in a multiflow scenario.

sending rate which will *anticipate* the first loss cycle. The system later evolves in a way similar to Fig. 3.3-(a), since after TCP halves its window, the capacity drains the queue empty and LEDBAT corrects its wrong base delay estimate. In subsequent cycles, LEDBAT will dutifully yield to TCP.

By means of Fig. 4.1, we show, instead, that the interaction among LEDBAT flows is heavily influenced by the buffer size B and the start time gap ΔT . Each graph reports the sender window of two competing LEDBAT flows. In the top plot, obtained for $(\Delta T, B) = (2, 40)$, the second flow activates before the first one has started to create queuing. So, the two flows measure the same base delay and set the same target, which they together reach soon. But the first flow, having started before, attains a larger congestion windows, and actually owns the biggest share of the queue.

Instead, extremely different dynamics can be observed for $(\Delta T, B) = (10, 40)$ in the middle plot. In this case the second flow starts later enough to allow the first one to create some queuing delay, in particular a delay τ_T equal to its target. For this reason, the second flow wrongly senses a base delay equal to τ_T , and consequently sets its target to twice this value. Therefore, the newcomer starts increasing its rate right away, while the first one senses a growing queuing delay and begins to slowdown until, slightly after $t = 20$ s, it finally reaches the minimum rate.

Afterwards, dynamics depend on the specific buffer size. The middle plot shows a case where the buffer cannot accommodate the target queuing delay of the second flow (as $B=40 < 2\tau_P=41.6$). In fact, around $t = 25$ s, the second flow causes a loss on the bottleneck link and consequently drops its sending rate. Afterwards, similarly to the TCP case, the capacity drains the queue empty, providing the second flows the chance to correct its wrong base delay estimation. Subsequently, flows appear to share much more fairly the bottleneck capacity.

The bottom plot, depicts instead the effect of a larger $B = 100$ buffer, able to absorb the extra delay of the second flow. Basically, since no loss occurs, the second flow reaches its target and then settles, leaving the first flow in starvation. Unfortunately this unfair state persists for a possibly long time (namely, due to route changes considerations, the Draft [79] imposes a reset of the base

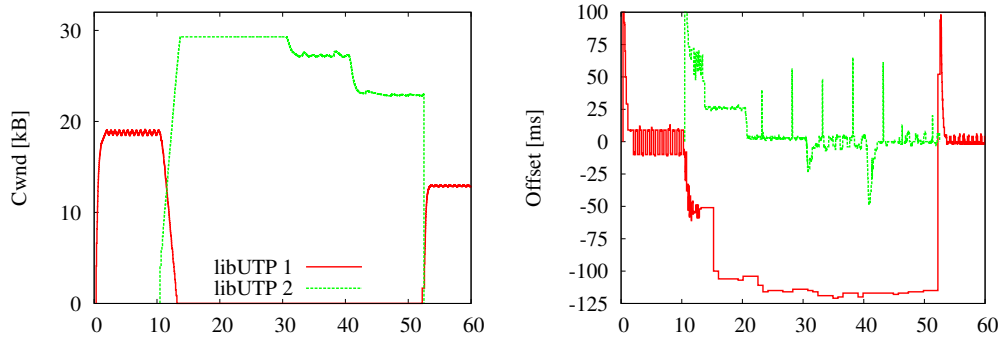


Figure 4.2: Experimental LAN testbed: Congestion window evolution (left) and offset from the target (right) for two competing backlogged libUTP flows.

delay every 2-10 minutes).

In Fig. 4.1-(b) we report that the same unfairness issue arise also in the multiple flow case, provided that the buffer is large enough to accommodate the outstanding data traffic. In that example we have 5 backlogged LEDBAT flows which start every four seconds and we can see that the last arrived flow takes all the bandwidth, starving the already established transfers, due to a wrong base delay estimation.

4.2.2 Latecomer in real networks

Similarly to the takeaway gathered via simulation, we point out that the latecomer unfairness also holds in practice, possibly leading to severe flow starvation. We show this by performing testbed experiments of the recently released BitTorrent open-source LEDBAT library [49] (named libUTP). As described above, we consider two PCs connected by a $C = 10$ Mbps Ethernet bottleneck, where we emulate by means of `netem` [50] a $RTT = 50$ ms delay. The first flow starts at time $t = 0$ while we let the latecomer join (and spoil) the party at $t = 10$ s. Backlogged transfers are started using the source code provided in [49], instrumented to produce detailed application-level logs¹. Results of the experiment are shown in Fig. 4.2, whose left portion reports the congestion window of the two flows over time. As soon as the first flow starts, it increases its congestion window until the target is reached, and then settles. However, when the latecomer kicks in at $t = 10$ s, the congestion window of the first-comer drops until starvation. The situation persists until $t = 50$ s, at which time we stop the latecomer transfer: right after, the first-comer opens its congestion window again, saturating the link. This is an unfortunate situation, that can however be easily corrected as we show in the following sections.

¹Packet level traces are also captured and post-processed as in Chap. 2 for cross-checking purposes: the results are in agreement with the application logs.

4.2.3 Current LEDBAT fairness issues

According to the original Draft proposal [79], LEDBAT maintains a minimum One-Way Delay estimation D_{min} , which is used as base delay to infer the amount of delay due to queuing. LEDBAT flows have a target queuing delay τ , i.e., they aim at introducing a small, fixed, amount of delay in the queue of the bottleneck buffer. Flows monitor the variations of the queuing delay $q(t) - D_{min}$ to evaluate the distance $\Delta(t)$ from the target:

$$\Delta(t) = (q(t) - D_{min}) - \tau, \quad (4.1)$$

where $q(t)$ is the queuing delay measured at time t . The value of the offset $\Delta(t)$ is then used to drive the congestion window evolution, which is updated packet-by-packet at each acknowledgement reception as follows:

$$cwnd(t+1) = \begin{cases} cwnd(t) + \alpha \frac{\tau - \Delta(t)}{\tau} \frac{1}{cwnd(t)} & \text{if no loss,} \\ \frac{1}{2} cwnd(t) & \text{if loss.} \end{cases} \quad (4.2)$$

where t is a discrete time variable that increments by 1 at each ack arrival and $cwnd(t)$ is the congestion window at time t . The drawbacks of such a congestion window update mechanism have been outlined in [28] and mainly consist in the intra-protocol unfairness coupled with a poor calibration of the LEDBAT level of aggressiveness with respect to TCP.

4.2.4 Impact of additive decrease

As proved in [28], the unfairness arising from two competing LEDBAT flows starting at different moments is due to the *additive decrease* component $\alpha \frac{\tau - \Delta(t)}{\tau}$ that intervenes when $\Delta(t) > \tau$. We thus argue that the additive decrease, rather than the measurement errors, is the main cause of unfairness in the LEDBAT protocol; in other words, the late-comer advantage is actually a fundamental drawback of the additive decrease term, meaning that the original design is currently misguided.

Without loss of generality, let us consider the case of N LEDBAT flows with the same round trip time $R(t)$, sharing the same link of capacity C and finite buffer size B . Each flow $i \in \mathcal{N}$, with $\mathcal{N} = \{1, 2, \dots, N\}$, starts at $t_i \geq 0$, with $t_1 \leq t_2 \leq \dots \leq t_N$ and with an initial congestion window W_i . Given the packet-level congestion window dynamics in (4.2), we demonstrate the following statement.

Proposition 1. *If $N < \frac{B}{\tau C}$, and $d_{max}(t_N) \triangleq \max_{i,j \in \mathcal{N}} [W^i(t_N) - W^j(t_N)] > 0$, then the system is unfair, i.e., $\exists t^* \geq t_N$, such that $\forall t > t^* d_{max}(t) > 0$.*

Proof. Given (4.2), a simple fluid representation of the window dynamics of flow i , $W_i(t)$, in continuous time, is:

$$\frac{dW_i(t)}{dt} = \frac{1}{R} \frac{\tau - Q(t)}{\tau}, \quad (4.3)$$

where we supposed for simplicity $R(t) \approx R$, which is true for large propagation delay (the proof can be easily extended to the case of variable round trip delays). Since the estimated queuing delay can be different for each flow, depending on its stored base delay, we replace $Q(t)$ by $Q_i(t)$, i.e., the queue occupancy measured by each sender, and simply observe that $Q_i(t)$ varies in the interval $(Q(t) - (N-1)\tau, Q(t))$. Indeed, the last flow makes the largest error in the estimation of the queuing delay, because it measures as base delay the actual propagation delay increased by $(N-1)\tau$, the sum of the target delay of all preceding flows. It follows that, $\forall i, j \in \mathcal{N}$:

$$W^i(t) - W^j(t) = W^i(t_N) - W^j(t_N) + \int_{t_N}^t \frac{Q_j(u) - Q_i(u)}{R\tau} du$$

where $|Q_j(t) - Q_i(t)|$ is bounded by $(N-1)\tau$. Hence, if we choose t^* equal to $t_N + \frac{W^{i^*}(t_N) - W^{j^*}(t_N)}{(N-1)/R}$, with $(i^*, j^*) = \arg \max_{i,j \in \mathcal{N}} W^i(t_N) - W^j(t_N)$, it results:

$$\begin{aligned} d_{\max}(t) &\triangleq \max_{i,j \in \mathcal{N}} W^i(t) - W^j(t) \\ &\geq \max_{i,j \in \mathcal{N}} W^i(t_N) - W^j(t_N) + \frac{(N-1)}{R}(t - t_N) \\ &= \frac{(N-1)}{R} \left((t - t_N) + \frac{W^i(t_N) - W^j(t_N)}{N-1} R \right) \\ &= \frac{(N-1)}{R}(t - t^*) > 0, \quad \forall t > t^*. \end{aligned}$$

□

Observation 2. *The fact that the system evolves towards an unfair state is strictly related to the fact that the dynamic equations, describing the state of the system are unstable. Besides equations (4.3) for the sources, we have*

$$\frac{dQ(t)}{dt} = \sum_{i=1}^N \frac{W_i}{R} - C \mathbb{1}_{Q_t > 0}. \quad (4.4)$$

Equations (4.3),(4.4) define a linear system of ODEs with characteristic polynomial $\lambda^{N-1} (\lambda^2 + NC\tau R)$. The corresponding eigenvalues are $\lambda_1 = 0$, $\lambda_{1,2} = \pm i\sqrt{\frac{N}{R\tau}}$, and the matrix associated with the system of ODEs is easily shown to be diagonalizable by standard algebra. As the eigenvalues have zero real part, the system cannot consequently be asymptotically stable. Being the matrix diagonalizable, the solution is limited for every t , however the dependence to the initial condition never vanishes because of the zero real part of the eigenvalues. In addition, the associated matrix cannot be inverted because of the zero eigenvalue which implies that the solution of the system has an

orbit around any (W_1, \dots, W_N, Q) such that $\sum_i W_i = RC$, $Q = \tau$. In other words, the linear response of LEDBAT is never able to make the stable point reachable from any initial condition: this is the root cause of the observed latecomer advantage phenomenon that we aim at solving in the following.

4.3 Proposed LEDBAT modification

To address the latecomer issue, we propose to modify the delay-based decrease term and *to introduce a multiplicative decrease* continuously driven by the estimated distance from the target, $\Delta(t)$. Intuitively, the multiplicative window reduction response to congestion allows the source sending rate to slow down enough to make a stable (and fair) point always reachable. Clearly, to guarantee at the same time fairness and protocol efficiency, a proper choice of the decrease factor has to be made, so as to prevent significant (and unnecessary) drops in the congestion window. In addition, we observe that the additive increase term as in (4.2) makes LEDBAT flows slow down the increase factor until the target τ is reached, in which case the window increase completely stops. This clearly implies a smaller convergence to the target and hence a minor efficiency if compared to the case of a constant additive increase factor independent of $\Delta(t)$. Based on the above observation, we propose to modify the increase term as well, and *to introduce an additive increase* according to a constant factor α as in TCP Reno. In this way, we expect to achieve better efficiency performance without violating the low priority requirements as expressed in the LEDBAT Draft. Indeed, by selecting $\alpha \leq 1$ the additive increase component can be made at most as aggressive as TCP. Summarizing the observation from the previous section, we propose to modify the congestion window evolution as follows:

$cwnd(t+1) =$

$$\begin{cases} cwnd(t) + \alpha \frac{1}{cwnd(t)} & \text{if no loss and } \Delta \leq 0, \\ cwnd(t) + \alpha \frac{1}{cwnd(t)} - \frac{\zeta}{\tau} \Delta & \text{if no loss and } \Delta > 0, \\ \frac{1}{2} cwnd(t) & \text{if loss.} \end{cases} \quad (4.5)$$

In the following sections we quantify the overall improvement deriving by such a congestion window update by means of both a *fluid model*, which provides a closed-form characterization of the stationary throughput and *simulations*, which allow the study of more complex scenarios. In the remainder of this chapter, we refer to the modified version of LEDBAT as fair-LEDBAT (fLEDBAT).

4.3.1 Fluid model description

In this section we develop a fluid model of the congestion window and hence of the transmission rate of one or more fLEDBAT flows, aimed at capturing first order system dynamics. The congestion window is now a continuous variable both in time and in space, $W(t)$ (remainder of the

Table 4.1: Notation

N	Number of fLEDBAT flows
C	Link capacity
$\{W^i(t)\}_{i=1,\dots,N}$	Congestion windows at time t
$\{X^i(t)\}_{i=1,\dots,N}$	Instantaneous rates at time t
Q_t	Queue occupancy at time t
α	Additive Increase factor
ζ	Multiplicative Decrease factor
R_t	Round trip time at time t
τ	Queuing delay target

notation is summarized in Tab.4.1). We consider the case of N fLEDBAT flows sharing the same link of capacity C and experiencing the same² Round Trip Time R_t . In addition, we make the following assumptions:

- The round trip time R_t is defined by the sum of twice the propagation delay, R , transmission delay $1/C$ and queueing delay $q(t)$. We further assume that the propagation delay is predominant, i.e., $R_t \approx R$.
- The queueing delay $q(t)$ is defined as ratio of the queue occupancy Q_t at time t divided by the link capacity C , i.e., $q(t) = Q(t)/C$. Thus, we assume that the queueing delay information *instantaneously* propagates to the sender, neglecting thus the delay in the feedback loop.
- We further assume that flows can correctly estimate the queueing delay, which is equivalent to $D_{min} = 0$.
- By Little's law, we assume that congestion windows and link rates are linked by $X_t^i = W_t^i/R_t$, $\forall i = 1, \dots, N$.

Still, the assumption that flows can correctly estimate the queueing delay may not hold in practice. As such, we expect that simulation results may show an offset with respect to the model predictions, which is due to this simplifying assumption. There are however two main reasons for which we believe this assumption, which makes the problem tractable, is also reasonable. First, additional mechanisms to enhance the delay estimation accuracy could be then adopted in order to ameliorate the overall protocol performance: this has been done in previous work [60], and is also part of the current BitTorrent effort [38] to reduce the measurement error and hence reinforcing our assumptions. Second, a more fundamental reason is that the characterization of protocol dynamics in absence of such estimation error is a necessary step in the fLEDBAT protocol design – as, even on simplistic settings, important properties of the protocol such as efficiency and fairness can be *proved* to hold with the help of a rigorous framework.

²Though the model generalizes to the case of heterogeneous RTT, for the sake of simplicity in this chapter we focus on the homogeneous case.

4.3.2 Fluid system dynamics

Let us consider the case of N fLEDBAT connections, whose congestion window evolves according to (4.5). The corresponding flow-level congestion window evolution is:

$$\frac{dW_i(t)}{dt} = \frac{\alpha}{R} - \frac{\zeta}{\tau} \left(\frac{Q(t)}{C} - \tau \right) \frac{W_i(t)}{R} \mathbb{1}_{W_i(t) \geq 0} \mathbb{1}_{Q(t) \geq C\tau}, \quad (4.6)$$

where we denote by $W_i(t)$ the instantaneous congestion window at time t for connection i in the fluid system. As we assume an approximately constant round trip delay, we replace R_t by R in (4.6). The instantaneous queue occupancy instead satisfies:

$$\frac{dQ(t)}{dt} = \sum_{i=1}^N \frac{W_i(t)}{R} - C \mathbb{1}_{Q(t) \geq 0}. \quad (4.7)$$

where, in other words, only the flow that exceeds the capacity creates queuing in the buffer. Thus, the instantaneous rate of connection i , $X_i(t)$, satisfies:

$$\frac{dX_i(t)}{dt} = \frac{\alpha}{R^2} - \frac{\zeta}{R\tau} \left(\frac{Q(t)}{C} - \tau \right) X_i(t) \mathbb{1}_{\{X_i(t) \geq 0\}} \mathbb{1}_{Q(t) \geq C\tau} \quad (4.8)$$

and (4.7) can be re-written as:

$$\frac{dQ(t)}{dt} = \sum_{i=1}^N X_i(t) - C \mathbb{1}_{Q(t) \geq 0}. \quad (4.9)$$

4.3.3 Main results

We now present the main results of this chapter: namely, the existence of a unique and globally stable solution. We also express, with closed form formulæ, the performance of the protocol at the equilibrium, proving its *efficiency* and *fairness* – which was our initial goal. Let us start by proving that the system admits a unique solution.

Proposition 3. *The system of ODEs (4.8)-(4.9) admits the unique equilibrium $P^* = (X_1^*, \dots, X_N^*, Q^*)$*

$$X_i^* = C/N, \quad i = 1, \dots, N \quad \quad Q^* = C\tau + \frac{N\alpha\tau}{\zeta R} \quad (4.10)$$

where X_i^* and Q^* denotes the stationary values of X_i and Q respectively.

Proof. We consider the stationary regime by the condition $(\dot{X}_i, \dots, \dot{X}_N, \dot{Q}) = (0, \dots, 0)$

$$\begin{aligned}
\dot{Q} = 0 &\Leftrightarrow \sum_i^N X_i^* = C, \\
\dot{X}_i = 0 &\Leftrightarrow 0 = \frac{\alpha}{R^2} - \frac{\zeta}{RC\tau}(Q^* - C\tau)X_i^*, \\
&\Leftrightarrow 0 = \frac{\alpha}{R^2} - \frac{N\alpha}{CR^2}X_i^* \Leftrightarrow X_i^* = C/N, \quad i = 1, \dots, N.
\end{aligned} \tag{4.11}$$

□

Then, the following proposition states that this unique equilibrium is also globally stable (see [82]).

Proposition 4. *The system of ODEs (4.8)-(4.9) is globally stable in P^* .*

Proof. Let us write $\mathbf{X} = (X_1, \dots, X_N)$, we consider the trajectories of the point $(\mathbf{X}, Q) \in \mathbb{R}_+^{N+1}$ driven by the ODEs (4.8)-(4.9). In the region $A = \{\mathbf{x}, q : 0 < q < C\tau\}$, the state equations simplify to

$$\begin{cases} \dot{X}_i &= \frac{\alpha}{R^2} \Rightarrow X_i = X_i(0) + \frac{\alpha}{R^2}t, \quad \forall i \\ \dot{Q} &= \sum_{i=1}^N X_i - C \Rightarrow \\ Q(t) &= Q(0) + (\sum_{i=1}^N X_i(0) - C)t + \frac{N\alpha}{2R^2}t^2 \end{cases} \tag{4.12}$$

Clearly, for any $(\mathbf{X}(0), Q(0)) \in A$, there exists a finite $t \geq 0$ such that $(\mathbf{X}_t, Q_t) \notin A$. This means that all points $(\mathbf{X}_t, Q_t) \in A$ are unstable. The unique equilibrium point P^* , calculated in Prop.3, is outside A . For $(\mathbf{X}, Q) \notin A$, the state equations become

$$\begin{cases} \dot{X}_i &= \frac{\alpha}{R^2} - \frac{\zeta}{CR\tau}(Q - C\tau)X_i \\ \dot{Q} &= \sum_{i=1}^N X_i - C \end{cases}$$

We now use the technique of the Lyapunov function to show that P^* is a stable point, i.e., we have to show that there exist a function V_t defined in a neighborhood of P^* , positively defined for $t \geq 0$, with orbital derivative negatively semidefinite (in which case, the solution P^* is stable in the sense of Lyapunov, see [82] Theorems 8.1-8.3). Outside A , we define the Lyapunov function by

$$V(\mathbf{X}, Q) = \sum_{i=1}^N (X_i - X_i^*) - \log\left(\frac{X_i}{X_i^*}\right) + \frac{\zeta(Q - Q^*)^2}{2RC\tau} \tag{4.13}$$

Clearly, $V(P^*) = 0$, $V(\mathbf{X}, Q) \geq 0 \forall (\mathbf{X}, Q) \notin A$, and

$$\dot{V}(\mathbf{X}, Q) = \sum_{i=1}^N (\dot{X}_i - \dot{X}_i \frac{X_i^*}{X_i}) + \dot{Q} \frac{\zeta(Q - Q^*)}{RC\tau} \quad (4.14)$$

$$\begin{aligned} &= \sum_{i=1}^N \frac{\dot{X}_i}{X_i} (X_i - X_i^*) + (X_i - X_i^*) \left[\frac{\zeta(Q - Q^*)}{RC\tau} \right] \\ &= \sum_{i=1}^N (X_i - X_i^*) \left[\frac{\alpha}{X_i R^2} - \frac{\zeta(Q - C\tau)}{RC\tau} + \frac{\zeta(Q - Q^*)}{RC\tau} \right] \\ &= \sum_{i=1}^N (X_i - X_i^*) \left[\frac{\alpha}{X_i R^2} - \frac{N\alpha}{CR^2} \right] \end{aligned} \quad (4.15)$$

$$= \frac{\alpha}{R^2} \sum_{i=1}^N (X_i - X_i^*) \left[\frac{1}{X_i} - \frac{1}{X_i^*} \right] = -\frac{\alpha}{R^2} \sum_{i=1}^N \frac{(X_i - X_i^*)^2}{X_i X_i^*}.$$

Therefore $\dot{V}(\mathbf{X}, Q)$ is negatively semidefinite for any ball including the equilibrium point P^* . This proves that P^* is an equilibrium globally stable as per [82]. \square

Once we know that the system has a unique globally stable equilibrium, we want to show what the *convergence rate* of the system in a neighborhood of the equilibrium is. This can easily be evaluated considering *local stability* properties of the system.

Proposition 5. *The system of ODEs (4.8)-(4.9) is locally stable in the equilibrium $P^* = (X_1^*, \dots, X_N^*, Q^*)$*

Proof. We write $(\dot{X}_1, \dots, \dot{X}_N, \dot{Q}) = (f_1, \dots, f_N, g)$, for $X_i > 0$, $Q > 0$ where f_i and g are defined as follows:

$$\begin{cases} f_i(X, Q) = \frac{\alpha}{R^2} - \frac{\zeta}{C\tau R}(Q - C\tau)X_i \mathbb{1}_{Q(t) \geq C\tau} & i = 1, \dots, N \\ g(X, Q) = \sum_{i=1}^N X_i - C \end{cases} \quad (4.16)$$

Linearizing the system of ODEs in P^* , and defining $\Delta X_i = X_i - X_i^*$, $\Delta Q = Q - Q^*$, and $\mathbf{Y} = (\Delta X_1, \dots, \Delta X_N, \Delta Q)$ we obtain $(\tilde{f}_1, \dots, \tilde{f}_N, \tilde{g}) = \dot{\mathbf{Y}} = A\mathbf{Y}$ where A is a $(N+1) \times (N+1)$ square real matrix defined as follows

$$A = \begin{pmatrix} -\frac{\alpha}{CR^2} & 0 & \cdots & 0 & -\frac{\zeta}{C\tau R} \\ 0 & -\frac{\alpha}{CR^2} & \cdots & 0 & -\frac{\zeta}{C\tau R} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 1 & 1 & \cdots & 1 & 0 \end{pmatrix}$$

The characteristic polynomial is then

$$\left(\lambda + \frac{\alpha}{CR^2}\right)^{N-1} \left(\lambda^2 + \frac{\alpha}{CR^2}\lambda + N\frac{\zeta}{C\tau R}\right)$$

whose roots have all real part negative. \square

Proposition 6. *The solution of the system of ODEs (4.8)-(4.9) converges to the global stable equilibrium P^* at a rate $e^{-\Theta t}$ with,*

$$\Theta = \frac{\alpha}{CR^2} \left(\frac{1 + \mathbb{1}_{\zeta \leq \zeta^*} \sqrt{1 - \zeta/\zeta^*}}{2} \right)$$

$$\text{and } \zeta^* = \frac{\alpha^2 \tau}{4NCR^3}.$$

Proof. We calculate the dominant eigenvalue of the matrix A , i.e., the eigenvalue with the real part with the smallest absolute value. \square

To conclude, we summarize our main findings in the following observation, expressing the results in terms of the expected performance of fLEDBAT.

Observation 7. *Prop. 3,4,5,6 prove that the designed protocol is efficient ($X^* = C$), and long term fair ($X_i^* = C/N$). In addition the queuing delay ($Q^*/C = \tau + \frac{N\alpha\tau}{C\zeta R}$) attains the target τ by an error of $\frac{N\alpha\tau}{C\zeta R}$.*

Thus, our initial goals of an *efficient* and *fair* protocol are met. Clearly, a number of issues need further investigation (i.e., how the protocol performs in practice where not all modeling assumptions hold, what the impact of parameters and of packet-level dynamics is, how it performs against TCP, etc.) that we investigate in the next section by means of a thorough simulation campaign in a number of different scenarios.

4.4 Simulation overview

So far we have developed a mathematical model of our new proposed protocol in order to formally prove its properties. However, the model is based on a number of simplifying assumptions and it neglects some aspects due to packet-level quantization (i.e., queue length and congestion window in multiple of fixed-size packets as opposed to continuous rate in the fluid model). Hence, in the remainder of this chapter we carry out a thorough packet-level ns2 [11] simulation campaign, to cope with scenarios where such assumptions do not hold. Our implementation is available as open source at [7]: besides, we point out that our code can be used as a ns2 module in simulation, or as a Linux kernel module for experimental studies³.

³Since the LEDBAT module is implemented using *integer arithmetic* only, it can be run as a kernel module; instead, as fLEDBAT employs *floating-point arithmetic*, for the time being it can only be used as a ns2 module.

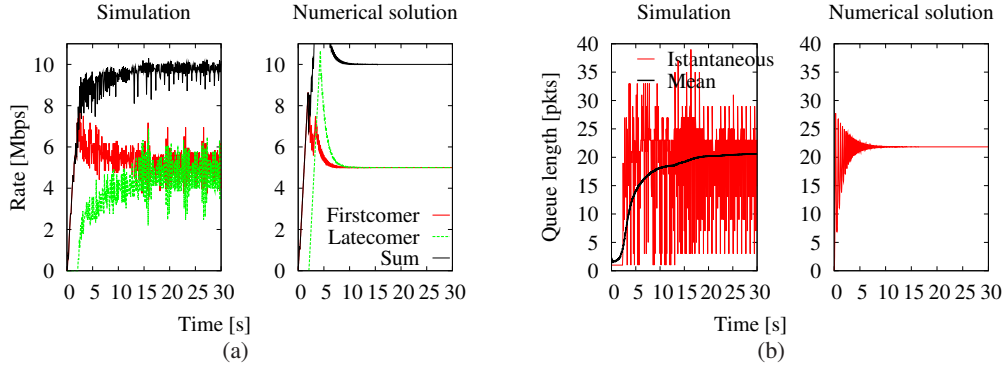


Figure 4.3: Comparison of (left) simulation and (right) numerical solution for (a) Rates and (b) Queue length. The similar average values in left and right plots confirm a good fit between packet level simulation results and flow level numerical results.

Unless otherwise stated, we consider a reference scenario consisting of a bottleneck link of capacity $C = 10$ Mbps and buffer size $B = 100$ packets (about 4 times the LEDBAT target). For the sake of simplicity, we consider fixed packet size equal to $P = 1500$ Bytes. Data flows in a single direction, and ACKs are not delayed, dropped nor affected by cross-traffic on their return path (except in the P2P scenarios reported in). All flows have the same round trip time $RTT = 50$ ms, half of which is due to the propagation and transmission delay components of the bottleneck link (i.e., a one-way base delay of 25 ms), to which we add a jittering component uniformly distributed in $[0,1]$ ms to avoid synchronization issues. We defer the study of more realistic scenarios, including heterogeneous delays, different access technologies, background traffic and P2P-like traffic models to Sec. 4.7. As far as TCP flows are concerned, we select the TCP NewReno flavor, enabling the selective acknowledgement SACK option due to its growing widespread use [66]. By selecting the TCP NewReno flavor, we gather conservative results since we expect more recent TCP variants implemented by default in Linux and Windows (respectively Cubic [77] and Compound TCP [80]) operating systems to be more aggressive than traditional TCP NewReno flows. Each simulation point reported in the following is the results of 10 simulation runs, over which we gather the average and standard deviation of the metrics under study.

However, we still need to provide evidence of the fluid model accuracy; we do so by comparing the numerical solution of the fluid model with `ns2` simulation results. We consider the simple network scenario described above, and two fLEDBAT flows with the same target ⁴ $\tau = 25$ ms. For the time being, we fix the decrease component by setting $\zeta = 0.1$ (and explore the impact of ζ

⁴Notice that delay target τ was initially set to 25 ms in the IETF Draft and to 100 ms in the BEP-29 specification. However, as shown in Chap. 3, provided that all flows have the *same target value*, which is furthermore set to a value *not exceeding the buffer size*, the actual value of τ is not critical. Hence, results shown in the following are valid for both target settings.

later on). To recreate the conditions for the latecomer unfairness phenomenon, the two flows do not start at the same time, but their start times are separated by a gap (2 seconds in the figure). The system state over time is depicted in Fig. 4.3, which reports both the flow rates X_1, X_2 (a) and the buffer occupancy Q_t (b) gathered either by numerical solution (right) or ns2 simulation (left). As a general comment, the numerical solution shows a good agreement with the simulation results (although as expected packet-level dynamic exhibits much wider fluctuations, while the fluid model gives an average behavior). Indeed, notice that the *average* of queue occupancy and flow rates yielded by the fluid system closely matches the simulation dynamics (for the sake of readability, we plot the *moving average* of the queue length gathered via simulation alongside the instantaneous occupancy). As expected, both numerical and simulation results show that the capacity is, after an initial transient phase, fairly shared among flows (i.e., $\bar{X}_i \approx C/2, \forall i$) and that furthermore the queuing delay target is reached (i.e., $\bar{Q}_t \approx C\tau$).

In the following we study several aspects of the LEDBAT protocol family, that we separately report to better isolate their effect. Sec. 4.5 addresses the impact of different *traffic models* on protocol performance, considering backlogged vs chunk-by-chunk transfers. Then, Sec. 4.6 addresses a *sensitivity analysis* of the protocol to ζ parameter variation. Finally, Sec. 4.7 compares LEDBAT and fLEDBAT under heterogeneous P2P-like scenarios.

4.5 Impact of traffic model

In this section we assess how the fLEDBAT protocol deals with different kinds of traffic. Besides the classical backlogged transfer, we simulate a chunk-based transfer, which mimics the behavior of a BitTorrent data exchange between two peers.

4.5.1 Chunk-by-chunk transfer

In this scenario, we consider sources that continuously transmit chunks of data, where each chunk has the typical BitTorrent size of 250 kB (nearly 170 full payload packets). As soon as a chunk transmission ends (i.e., when the last acknowledgment for that chunk has been received at the sender side), a new chunk transmission is scheduled with the same peer. This traffic model, which emulates the dynamics of P2P traffic exchange, differs from backlogged transfers in that, after the last data packet of a chunk has been sent, the source peer stops transmitting for about RTT seconds until the matching acknowledgement is received, and a new chunk transmission can start (i.e., chunks transmission is *not* pipelined). Notice also that we keep the congestion window parameter across chunks (i.e., the congestion window is *not* reset between subsequent chunks exchanged with the same peer).

Fig. 4.4-(a) reports the time evolution of the system dynamics when $\zeta = 0.01$: in the left portion, congestion window and base delay estimation of the firstcomer (top) and latecomer (bottom) flows are reported, while the right portion shows the distribution of the queue length. We can see that, despite the latecomer initially has an incorrect view of the base delay (as in LEDBAT), the multiplicative decrease phase of the firstcomer allows the latter to correct its estimate, after which

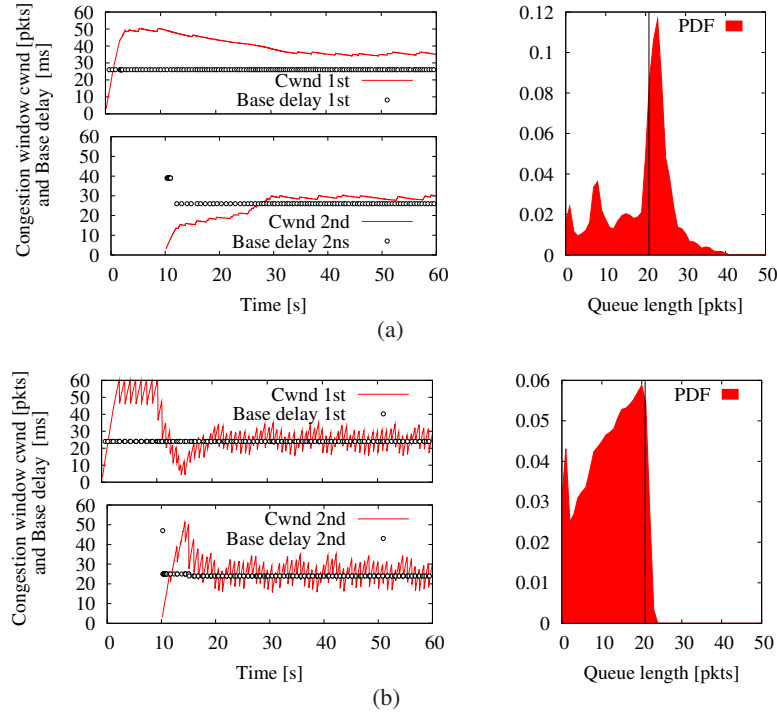


Figure 4.4: Time evolution of fLEDBAT dynamics with (a) chunk-based $\zeta = 0.01$ and (b) backlogged $\zeta = 5$ traffic models.

the performance share converges to an equitable state. Due to (i) the continuous adjustment of AI and MD dynamics and (ii) the fact that chunk transmission seldom pauses the transmission, the queue is no longer stable as for the standard LEDBAT case described in Sec. 4.5, but fluctuates around the occupancy value predicted by the model (represented by a solid vertical line).

4.5.2 Backlogged transfer

In the case of backlogged transmission, a latecomer phenomenon may still arise depending on the value of ζ : indeed, when ζ is too small, the multiplicative decrease component of the first flow is slower than the additive increase of the latecomer, which is thus unable to correct its wrong estimation. However, provided that ζ is large enough to let the queue flush, fLEDBAT can still reintroduce fairness.

Results for the backlogged scenario are reported in Fig. 4.4-(b) for $\zeta = 5$. Especially, the queue now seldom flushes, as it can be seen by the increased probability to have a null queue length shown by the PDF. In turn, this helps latecomers gain a correct view of the base delay. In this case though, the model slightly overestimates the queue size: indeed, due to larger ζ values,

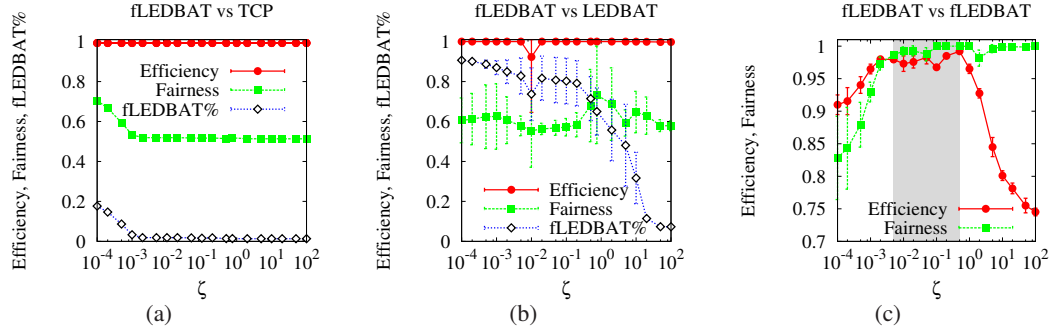


Figure 4.5: Sensitivity analysis to ζ : Efficiency, long-term fairness and protocol breakdown of a fLEDBAT flow sharing the bottleneck with (a) a TCP flow, (b) a LEDBAT flow (c) another fLEDBAT flow.

the congestion window fluctuations are now wider. Also, as the queue flushes, the protocol is less efficient with respect to the previous cases, because the capacity is not fully utilized all the time.

We point out that, since fLEDBAT is designed to be a low-priority protocol, slight inefficiency can be tolerable if they reintroduce correctness of operation in general settings—especially if otherwise this could have serious consequences, as in the case of a latecomer long-lived backup starving the firstcomer.

4.6 Sensitivity analysis

In this section we carry out further simulations to assess the impact of the choice of the parameter ζ on the protocol performance. In order to gather a complete sensitivity analysis of fLEDBAT parameters, we consider several scenarios: (i) a TCP Reno flow competing with a fLEDBAT flow, (ii) a LEDBAT flow competing with a fLEDBAT flow, (iii) two fLEDBAT flows competing at the same bottleneck. All flows operate in chunk-by-chunk transmission mode.

As performance metrics, among the ones already defined in Sec. 3.2.2, we consider *fairness* (η) and *efficiency* (F). For the sake of illustration, we also consider the *protocol breakdown*, defined as the percentage of traffic sent by fLEDBAT sources over the total traffic, which immediately conveys the level of low priority of fLEDBAT with respect to other protocols competing at the same bottleneck.

4.6.1 fLEDBAT vs TCP

Fig. 4.5-(a) shows the efficiency and fairness performance when a single TCP and a single fLEDBAT flow share the bottleneck; first of all, we can see that low-priority goal is met, as TCP is enjoying the largest portion of the capacity (fLEDBAT breakdown goes to 0% and fairness drops

to $1/N$). As expected, efficiency is high: as we already observed in Sec. 3.2.5 for LEDBAT, fLEDBAT is still able to push some bytes on the link, thereby increasing the overall link utilization with respect to the case where a single TCP Reno pass through the bottleneck.

With the exception of extremely low values of $\zeta < 10^{-3}$ (which soften the effect of the multiplicative decrease, and sharpen the impact of the TCP Renolike additive increase), the low-priority goal is therefore satisfied. Thus, selecting ζ is not a concern as far as heterogeneous fLEDBAT vs TCP scenarios are considered.

4.6.2 fLEDBAT vs LEDBAT

Fig. 4.5-(b) shows the efficiency and fairness performance when a single fLEDBAT flow and a LEDBAT share the bottleneck. We randomize the start time of both flows in the $[0,10]$ sec interval, so that the latecomer can be either of the two protocols. In this case, we infer that fLEDBAT is generally more aggressive (due to the AI dynamic) until ζ grows too large, in which case the reverse happens (due to the MD dynamic). Specifically, less than 20% of the bottleneck is occupied by LEDBAT when $\zeta < 10^{-2}$. For larger values of ζ though, LEDBAT becomes increasingly competitive with fLEDBAT: the crossover happens at about $\zeta = 5$, after which fLEDBAT becomes even lower priority than LEDBAT.

In no case, however, the share is fair and a latecomer phenomenon may still arise. Consider that, when LEDBAT starts first and saturates the bottleneck, it induces a very steady queue. Therefore, when an fLEDBAT latecomer flow arrives at the bottleneck, it measures an incorrect base delay. However, as LEDBAT reacts with a *linear* decrease to the increasing delay, the fLEDBAT latecomer will not have the opportunity to correct its estimate – as it otherwise does whenever the firstcomer flow reacts with a *multiplicative* decrease to the increasing delay. Hence, when ζ is small, the fLEDBAT latecomer can starve the LEDBAT flow.

4.6.3 fLEDBAT vs fLEDBAT

We finally consider the intra-protocol scenario in which two fLEDBAT flows share the bottleneck. We set the start time of latecomer flow to $t = 10$ s, which was shown in Sec. 3.2 to represent a worst case scenario for the fairness index. Fig. 4.5-(c) reports results for varying ζ , focusing on efficiency and fairness metrics. From the figure, it is clear that fLEDBAT is able to operate fairly and efficiently under a wide range of parameters. Overall, taking into account also the previous remark in the intra-protocol fLEDBAT vs TCP scenario, we have that any value of ζ in the gray shaded zone yields to an efficient, fair and low-priority system.

4.6.4 Tuning ζ for multiple-flows scenario.

In this scenario we present results for a varying number of fLEDBAT flows competing at the bottleneck, with in $N \in [2, 10]$. Results are reported in Fig. 4.6, where we select a few values of $\zeta \in \{0.01, 0.1, 0.2, 0.5\}$ from the shaded gray zone of Fig. 4.5. As can be seen, it is always possible to find a value of ζ that guarantees fairness for the whole set of flows, with any values in the

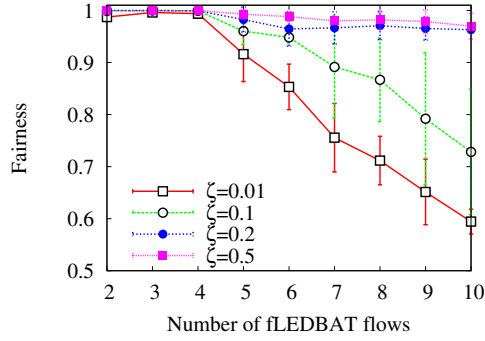


Figure 4.6: Variation of the intra-fairness index of fLEDBAT, for different values of ζ and a growing number of flows.

range providing good results for the number of flows that are typically concurrently active in BitTorrent. Moreover, the very same values of ζ that provide fair resource share, were already shown to provide efficient use of the resources for $N = 2$ flows in Fig. 4.5-(c), which clearly holds for $N > 2$ (omitted to avoid cluttering the pictures).

4.7 Appetizer to P2P scenarios

In order to compare fLEDBAT vs LEDBAT performance under other conditions, closer to the BitTorrent use-case, we finally consider a chunk-based scenario that (i) loosely mimics the behavior of BitTorrent peers, (ii) employs Internet-like heterogeneous delays and access rates, (iii) considers background traffic and coupled queues, (iv) addresses the impact of chunk sizes.

We consider two different scenarios: first a single BitTorrent peer and a single queue in isolation, using a more sophisticated traffic model and observing the effect of delay heterogeneity alone. Second, we study a BitTorrent swarm-like scenario, as close as possible to the actual target application scenario of LEDBAT, with 100 peers continuously exchanging data among them. In this case, the state of queues is no longer independent, as data traffic mixes with acknowledgement traffic in the reverse direction, causing a coupling of the congestion controllers as well. To gather even more realistic results, we finally add HTTP-like background traffic to the swarm scenario, studying its impact on congestion controls dynamics.

Before presenting the results of our simulations, it is worth saying a few words on the traffic model we use to simulate a BitTorrent peer. Like a real BitTorrent source [37], our simulated peers open a number of connections to other N peers, but they actively exchange chunks of data with only a restricted number $M < N$ of the available peers at the same time (set to $M = 5$ and $N = 10$). We employed different chunk sizes, ranging in $[250 - 4096]$ KB [64], using 250 KB chunks unless otherwise stated. At the end of each chunk transmission, the sender chooses the next destination peer as follows: with a *persistence probability* P_P , the sender will send another chunk

to the same peer, keeping the congestion window settings; with probability $(1 - P_P)$, the sender will choose an inactive neighbor at random, resetting in this case the congestion window to 1. A detailed discussion of the meaning of different values of P_P is found later on.

We point out that the goal of the study reported in this chapter is not on assessing the impact of LEDBAT on BitTorrent, that will be the subject of the second part of this thesis. Hence, we chose not to implement all the application-level details of a BitTorrent source (e.g., tit-for-tat, optimistic unchoking, signaling, etc.). Rather, our objective is assessing that fLEDBAT does not worsen flow-level performance with respect to LEDBAT: hence, we argue that this simpler traffic model is a good approximation of the actual peer interaction given our purpose.

As far as network conditions are concerned, we consider both *FTTH symmetric* access capacities ($C = 10$ Mbps, $B = 100$ packets, default unless otherwise stated) and *ADSL asymmetric* capacities ($C = 1$ Mbps uplink, $C = 8$ Mbps downlink $B = 100$ packets). To add further realism, we simulate both a *homogeneous* network setup (i.e., in which all peers have the same propagation delay $RTT = 50$ ms) as well as *heterogeneous* scenario (default unless otherwise stated) where the propagation delay of the access link of each peer is distributed according to realistic delay measurement [83], with mean equal to 37.9 ms.

4.7.1 Single peer perspective

Let us start by considering a single peer behind a FTTH connection, exchanging 250 KB chunks with peers chosen according to the algorithm described above: when a chunk transfer is completed, the source keeps the same destination peer with a probability P_P , and changes it with probability $(1 - P_P)$. fLEDBAT and LEDBAT are simulated separately, i.e., all peers either use the former or the latter protocol, and we consider both an homogeneous and heterogeneous delay scenario. For fLEDBAT we set the parameter $\zeta = 0.1$, which yielded good performance in the sensitivity analysis.

In our experiments we explore the full range of $P_P \in [0, 1]$. As $P_P \rightarrow 0$, we expect the performance of the two protocols to be close: indeed, when connections are reset every 170-th packet (which corresponds to 250 KB chunks), the protocols are basically always in transient state and the target is not even likely reached during a chunk transfer. Conversely, differences are expected to arise in the more stable scenarios $P_P \rightarrow 1$, where congestion parameters are kept across chunks. Actually, we expect a real BitTorrent source to have a behavior similar to a sender with $P_P \geq 0.8$: in fact, one source normally tries to keep 4 out of the 5 “best” (i.e., higher capacity) peers while at the same time continuously discovering new, potentially “better”, peers (i.e., by means of optimistic unchoking).

Results of the comparison are reported, in term of efficiency and fairness, in Fig. 4.7. Since flows are no longer backlogged, from now on we consider *short-term fairness*, measured at 1 Hz rate (i.e., corresponding to a good tradeoff between a minimum number of RTTs to have statistically meaningful results, and a maximum time lag, as flows may die out after a single chunk transfer). Notice that we expect short-term fairness to be harder to achieve than the long-term fairness considered in the previous sections (hence we expect lower F values).

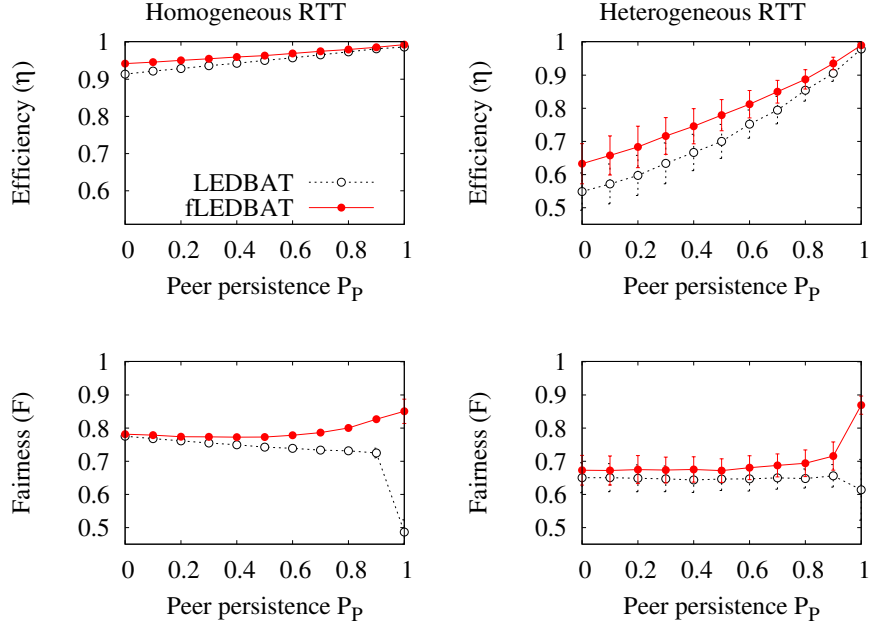


Figure 4.7: Efficiency (top) and Fairness (bottom) of fLEDBAT in the homogeneous (left) and heterogeneous (right) RTT scenarios.

At first glance, we remark that under all scenarios and P_P values, fLEDBAT is more efficient (due to AI) and fair (due to MD) with respect to LEDBAT, and the gain is more evident exactly in the operational range of BitTorrent (i.e., $P_P \geq 0.8$). In the homogeneous case depicted in the two plots on the left, as expected, the fairness gap exacerbates as $P_P \rightarrow 1$: in this case, fLEDBAT's ability to correctly measure the base delay leads to an increase of the fairness metric. On the contrary, LEDBAT fairness decreases as P_P grows, due to the latecomer issue: the effect is stronger when $P_P = 1$, as in this case the unfair situation persists through the whole duration of the experiment and leads to a consistent drop of F . Similar considerations hold for the heterogeneous case (see plots on the right), although in this case the heterogeneous delays introduce RTT unfairness in both fLEDBAT and LEDBAT, reducing the absolute value of F (i.e., we do not attempt to account for the RTT bias in the fairness definition). However, RTT unfairness does not translate into serious issues (such as long time starvation), and fLEDBAT always guarantees a fairness higher than LEDBAT (as latecomer advantage disappears).

As far as efficiency η is concerned, when the congestion window parameters are reset at every chunk transmission ($P_P = 0$), the link capacity is not fully utilized even in the homogeneous case. The heterogeneous case further adds inefficiency, as flows with higher RTT increase their congestion window more slowly, hence further wasting link capacity. However, it is worth pointing

out that the additive increase component of fLEDBAT makes it more efficient than LEDBAT under any circumstance, while the multiplicative decrease component guarantees at the same time its lower-priority with respect to TCP.

4.7.2 Entire swarm perspective

We now consider an entire swarm, where 100 peers continuously exchange data among them using 5 parallel upload slots, as in BitTorrent: hence, the uplink queue of each peer contains a mix of (i) data packets being uploaded to the swarm and (ii) acknowledgement packets related to data being downloaded from the swarm. As this happens for each queue, P2P traffic interacts both in the forward and backward directions. Notice also that the end-to-end congestion controls of data transfers are tightly coupled in a non-trivial way, as in our model each peer maintains multiple connections to a set of other peers, which moreover dynamically evolve over time. We do not try to model all BitTorrent dynamics (e.g., chunk trading logic), but rather assumes that peers are always able to find content where they seek it (i.e., a large file where there is enough chunk diversity in the system). Therefore, we do not attempt at measuring application-level statistics, such as the torrent download time, but rather focus on the transport-layer fairness statistics.

Unlike the previous scenario, here we fix the persistence probability to $P_P = 0.8$. As shown in the former experiment, each interruption in data transmission favors LEDBAT, for the consequent draining of the queue lets the protocol correct the base delay estimation. Under this consideration, since BitTorrent optimistic unchoking happens on 1 slot out of 5 at low rate (each 30 seconds), $P = 0.8$ corresponds to a lower bound (since the chunk transfer can be much shorter than 30 seconds) and gives an optimistic (thus, conservative) estimate for LEDBAT fairness. Moreover, our simulation model also introduces an *idle RTT* (i.e., time elapsed between the transmission of the last data packet of a chunk and the reception of its acknowledgment) before a new chunk is sent to a persistent peer, that actual BitTorrent systems avoid by means of request pipelining and that thus further simplifies the LEDBAT task. However, we can partly compensate for this effects by employing larger chunks.

Since homogeneous and heterogeneous RTT scenarios yielded qualitatively similar results, in the following we only consider heterogeneous delay settings for more realism, with either FTTH or ADSL access. Finally, we also simulate a scenario where peers browse the Web while participating in the swarm: in particular we add an HTTP source from which peers download Web-pages (files with a size uniformly distributed in the range $[0 - 512]$ KB) using traditional TCP. There are always 25 peers, selected at random, with active HTTP downloads, so that a quarter of the total swarm is always involved in short interactive background Web transfers.

Short-term fairness results are shown in Fig. 4.8: first, notice that the coupling of queues is more beneficial for the fairness index (for both LEDBAT and fLEDBAT) with respect to the single queue scenario shown in Fig. 4.7. At the same time, in all cases fLEDBAT consistently achieves higher fairness than LEDBAT. Larger chunk sizes, as expected, badly affect LEDBAT, because they reduce the number of transmission interruptions (which helped in emptying the queues). Conversely, fLEDBAT appears almost insensitive to chunk size.

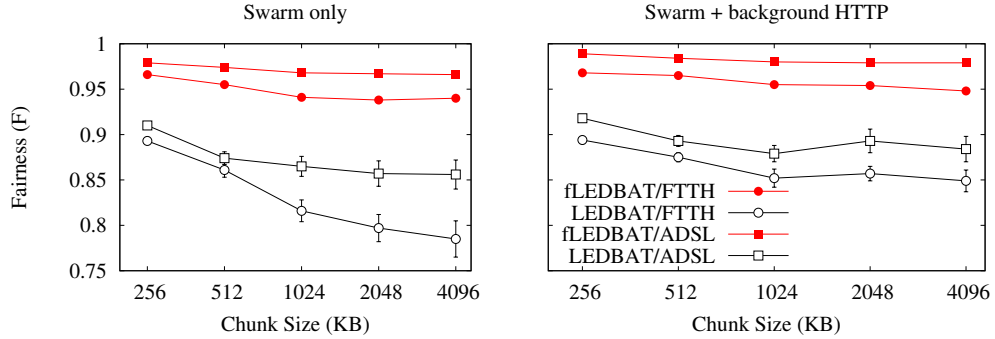


Figure 4.8: Fairness in FTTH and ADSL swarm-like scenarios, with and without background HTTP traffic.

Comparing ADSL with FTTH scenarios, we see that in the latter case a further cause of unfairness may arise: indeed, as capacities are symmetric and one peer may have several downloads ongoing (only upload slots are limited in number), downlink can become a bottleneck as well (e.g., in real systems this may happen in case of popular torrents with many seeds). The impact of background HTTP traffic is instead beneficial to the fairness of both fLEDBAT and (especially) LEDBAT: this is due to the fact that peers downloading HTTP data will send out a burst of acknowledgement packets, that possibly cause buffer overflows in the uplink queues (which assist in raising fairness in LEDBAT, as shown in Sec. 3.2). Background traffic has instead a smaller impact on fLEDBAT, as the protocol achieves higher fairness without external help.

4.8 Summary

In this work, we proposed modifications to the LEDBAT congestion control algorithm that not only are able to achieve *low-priority inter-protocol* (i.e., against TCP) and *efficiency intra-protocol* (e.g., with other fLEDBAT flows), but also reintroduce *intra-protocol fairness*, solving thus the late-comer issues of the original LEDBAT proposal.

To model the protocols dynamic we used a fluid-model approach which allows, on the one hand, to detect the main issue at the base of LEDBAT unfairness (i.e., the additive decrease component) and on the other hand, to prove the correctness of fLEDBAT design. We also derived closed-form expressions for the average rate and queue length in the general case with N sources. Furthermore, by means of packet-level simulations, we further assess that fLEDBAT can safely operate under a number of scenarios (such as chunk-by-chunk and backlogged transmission) as it is not sensitive to parameter selection and operates reasonably well under a wide range of parameters. Finally, we tested the protocol in multiple-flows P2P-like realistic scenarios with heterogeneous RTTs and transfer emulating the operations of a BitTorrent peer, the original target application for LEDBAT.

Our simulations confirm the soundness of our proposal. Indeed, fLEDBAT not only solves the

fairness issue for backlogged flows, but maintains the same good properties of LEDBAT— that is, it yields to TCP while exploiting the spare bandwidth. Overall, we see that our proposed modifications lead to a demonstrable improvement in performance with respect to LEDBAT, in terms of both fairness and efficiency. Our simulations confirm fLEDBAT robustness even under realistic heterogeneous network conditions, on which BitTorrent can be expected to operate.

Apart from the intra-protocol unfairness, which was solved in the general case this work, in our opinion there is still a critical point in the LEDBAT algorithm definition that remains an open issue. This issue, described in Sec. 3.2.4 as well as in [78] and debated in the LEDBAT IETF working group mailing list, concerns the use of a *fixed* queuing delay target. Such fixed settings (which are referred to as “magic numbers” in the mailing list) are indeed not a good practice, as they may lead to undesirable behavior: as a matter of fact, not compliant implementation may set a higher target with respect to the mandatory standard values (i.e., malicious backlogged users can hence easily obtaining an unfair advantage over compliant users). Furthermore, a fixed queuing delay target is not scaling with the link capacity: i.e., while 100 ms queuing may be reasonable for todays ADSL modem (with large buffer size relatively to their narrow uplink capacities), this will not scale when high capacity Fiber-to-the-Home (FTTH) access will be ubiquitous. However, while fixed settings are not robust against malicious or misconfigured implementations, at the same time there is no obvious way of defining an *adaptive* target without loosing a bounded guarantees on the additional delay, that interactive applications would like to keep as small as possible.

Finally, while LEDBAT protocol has been also specified as an IETF protocol, we have to keep in mind that its main application is BitTorrent. Hence we carry out the rest of the analysis in Part. II using LEDBAT, and not fLEDBAT.

Part II

BitTorrent swarm perspective

Chapter 5

Impact of LEDBAT on BitTorrent - Simulative approach

So far we focused our attention on the investigation of the LEDBAT congestion control behavior on rather simple settings (i.e., single bottleneck, few backlogged flows, etc.), that are necessary to unveil the performance from a flow perspective, but that are also fairly far from the P2P settings in which the protocol is deployed. Moreover, prior analysis typically addressed questions, such as fairness and efficiency, that are natural from a congestion control context perspective, but are not directly related with the performance perceived by the users of the overall P2P system.

The main aim of LEDBAT is indeed to fully exploit the access capacity while introducing only a *low extra queueing delay on the end-to-end path* [79]. In the context of the popular file-sharing application, its goal is perhaps better understood in BitTorrent's developer words "*The main purpose of uTP is not to increase transfer speeds, it's to fix the problem of delay. [...] That said, the fact that uTP doesn't add significant delay also means that utilization (and speed) can be increased.*" [72] Also, it is clearly understood that users will be the ultimate judge of the application performance, as observed by Simon Morris, BitTorrent Product Manager "*unless we can offer the same performance [of TCP], then people will switch to a different BitTorrent client.*" [69].

In this chapter, we refine the understanding of LEDBAT, by gauging its impact on the primary BitTorrent user-centric metric, namely the torrent download time, by means of packet level simulation on ns2. An experimental study on the same issues in real-world networks is presented in Chap. 6. Results of our initial but careful investigations show that: (i) in case LEDBAT clients fully substitute TCP clients, no performance difference arise; (ii) in case of heterogeneous swarms, comprising peers using LEDBAT and TCP congestion control, completion time of LEDBAT peers can possibly benefit of lower uplink queueing delays, as signaling traffic (e.g., chunk requests) are not slowed down by long waits in the ADSL buffers.

The rest of this chapter is organized as follows. Sec. 5.1 describes the adopted methodology, as well as the scenario and parameter settings for the simulations environment. In Sec. 5.2 we present the main results of our investigation, varying the seeding strategy and swarm population, as well as

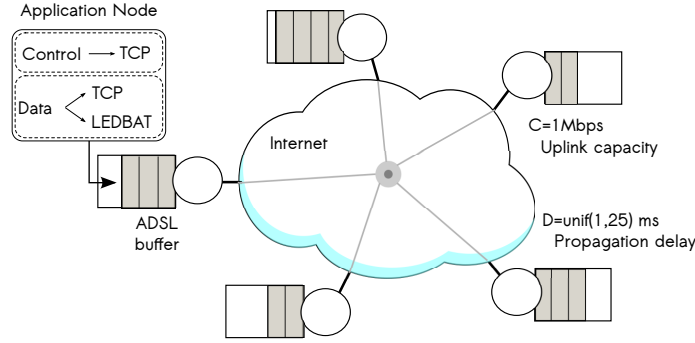


Figure 5.1: Synoptic of simulation model.

the target settings for the peers within the same swarm. Lastly, Sec. 5.3 draws the conclusions of the chapter.

5.1 Methodology and Scenario

To carry out our study, we resort to the open-source implementations of BitTorrent [2, 41] and LEDBAT (introduced in Chap. 3) for `ns2`. Nevertheless some modifications were necessary in order to successfully integrate both modules, which we shortly summarize in the following with the help of Fig. 5.1.

As commonly assumed, we consider the bottleneck to be represented by the access link, which is also one of the main motivations of LEDBAT. As access links technology, we consider ADSL-like connections with $C = 1$ Mbps uplink capacity (homogeneous for the whole peer population) and 8 Mbps down link capacity. Coherently with [78], unless otherwise stated, we set the buffer size to $B = 200$ full-size packets (corresponding to a few seconds given C). Access links also model propagation delay, which is chosen uniformly at random in the interval $[1, 25]$ so that the average one way delay of the swarm is about $D = 25$ ms (i.e., as two access links are crossed in the end-to-end path connecting any two peers). The target delay for the LEDBAT protocol is fixed at $T = 100$ ms, which is the default value in the BEP-29 specification [71] and later version of IETF Draft [79]¹ Access modems are then interconnected directly to the Internet, that we model by means of infinite capacity, null delay links connected through an (infinite switching capacity) router, to which all ADSL modems are interconnected in a star topology configuration.

We implement two kind of P2P nodes, that model different application settings as far as the congestion protocol used for *data transfer* is concerned. We assume that the latest application versions will by default initiate data transfers using LEDBAT on their uplink, but that they can accept

¹An initial set of analysis was performed using a target equal to 25 ms, as was specified in early versions of the LEDBAT Draft, leading to similar final consideration. For this reason, only the results with $T = 100$ ms are reported in this dissertation.

incoming TCP data connections (i.e., equivalent to setting `bt.transp_disposition=5` in μ Torrent); similarly, older versions initiate TCP transfers, but accept incoming LEDBAT connections anyway (i.e., `bt.transp_disposition=10`). We further model that, irrespectively of their data transfer settings, applications will use TCP for the exchange of *control messages*. Hence, a different traffic mixture will possibly compete for ADSL access link capacity and buffer space. Notice that, while nodes are free to decide what congestion control flavor use for the outbound connections, they have to comply to other peers settings as far as inbound data connections are concerned.²

We simulate a (mild) flash-crowd scenario, in which at time $t=0$ the swarm is constituted by only one seed, and then 100 leechers join with exponentially distributed arrival times (mean rate 0.1 Hz). During each simulation, we discard the first 50 completion samples that happen during the transient period (more details later on), and consider only the subsequent 50 completion times. For each parameter settings, we repeat each simulation 10 times, so that statistics represent 500 individual torrent downloads per setting.

As far as the swarm population is concerned, we either consider *homogeneous swarms* (i.e., all TCP or all LEDBAT) or *heterogeneous swarms* where the population is equally split, on average, among LEDBAT and TCP peers (denoted in the following as 50-50). While we reckon that it would be interesting to explore different TCP vs LEDBAT ratios, we believe this split ratio to be reasonable: while μ Torrent client is among the most popular clients, there is no precise agreement on its popularity (e.g., μ Torrent estimated share varies between 15% in 2006 to 60% in 2008 [86]), meaning that TCP versions are still around. As a side effect, the population sets size are not biased, thus the analysis of the results is simpler. Finally, recall that 50% of the traffic goes over LEDBAT as we saw in Fig. 1.1.

As far as the peer and seed behavior is concerned, we consider three different scenarios, namely (i) *never leave*, (ii) *random time*, and (iii) *immediate leave*.

In the first one, peers *never leave* the system after becoming seeds, thus continuing to serve other leechers in a optimistic swarm configuration. In the second, more realistic, scenario newborn seeds stay in the system for a *random time*. Under this conditions, each time a peer leaves, a new leecher joins the swarm, so that the swarm size remains constant. Specifically, peers stay in the system after becoming seeds for an exponentially distributed time, with mean equal to half their download time (this choice roughly models the BitTorrent netiquette, i.e., to continue seeding until the uploaded data reaches 1.5 times the downloaded amount). Finally, a worst-case scenario is considered, in which selfish peers *immediately leave* the swarm after data completion, while new leechers join to maintain a constant population in the torrent swarm.

5.2 Main results

In this section we report on the most relevant findings gathered using the network simulator, focusing on the impact of different settings for the swarm composition and seed strategy, as well as

²However, as we will see in Chap. 6 this is not how μ Torrent actually works.

target heterogeneity among different LEDBAT peers.

5.2.1 Swarm population and seed behavior

Simulation results are reported in Fig. 5.2. Top plots refer to scenarios where seeds never leave the system, middle plots to random stay scenario and bottom ones to immediate leave case. Taking a single run as an example, plots on the left depict how the completion time evolves during the simulation, ordered by peer completion rank (dark gray background represents the discarded initial transient period). If seeds stay forever, completion times shrink down to a point in which leechers are close to fully exploit their downlink capacity, which is no longer the case when seeds stay only for a finite time or leave as soon as they end the download.

Middle Fig. 5.2 plots report the completion time cumulative distribution (CDF) for the different populations. If peers never leave the system, no difference arises due to the congestion control algorithm: intuitively, as there is no resource hotspot, peers are able to download from many seeds at the same time, hence we do not expect congestion to play a major role in this case. Conversely, when seeds leave the system and are replaced by new leechers, resources become rare, translating into longer completion times. Notice that completion time in random stay scenarios is affected by the specific congestion control mechanism adopted by peers in case of heterogeneous swarms (while performance of homogeneous swarms are alike). Similar observations can be drawn for the *immediately leave* case, in which however the completion time of the heterogeneous swarm differs more when compared with the two homogeneous ones.

Reasons why this happens can be better understood by looking at the queue size complementary cumulative distribution (CCDF) shown in the right Fig. 5.2 plots (gathered by sampling all uplink queues at 10 Hz). Notice indeed that uplink queue of LEDBAT peers is very similar in all three cases, as LEDBAT tries not to exceed a target delay: deviation from target are due to TCP control connection sharing the same queue, and latecomer advantage, presented in Chap. 4. On the contrary, TCP queues can grow long: in scenario (b) and (c), about 20% of the cases, queues exceed 100 packets, corresponding to more than a second of queuing delay considering full size packets (as reported in the top x-axis for reference). In the 50-50 case, queuing delay aggregates both LEDBAT and TCP uplink buffers, resulting in an intermediate average system queueing time (notice that in the 50-50 scenario, a further deviation from LEDBAT target is due to bursty uplink ACK traffic of TCP connections opened by other peers in the swarm).

Interestingly, the completion time behavior changes if we consider a *heterogeneous swarm*, as shown in the middle plot of Fig. 5.2-(b). Investigating further, we find that LEDBAT peers have shorter download times in the 50-50 random-stay scenario, as the LEDBAT vs TCP breakdown of the completion time CDF of Fig. 5.3-(a) testifies (while no difference arises in the heterogeneous 50-50 never-leave scenario). This is a counter-intuitive result, as we would not expect completion time to be tied to the congestion control used to handle chunks upload: indeed, the completion time metric relates to the *download* performance of a LEDBAT peer \mathcal{P} (i.e., a peer using LEDBAT for its *uplink*), that rather depends on the uplink performance of other LEDBAT and TCP peers of the swarm (i.e., peers that upload to \mathcal{P} according to the protocol of their choice).

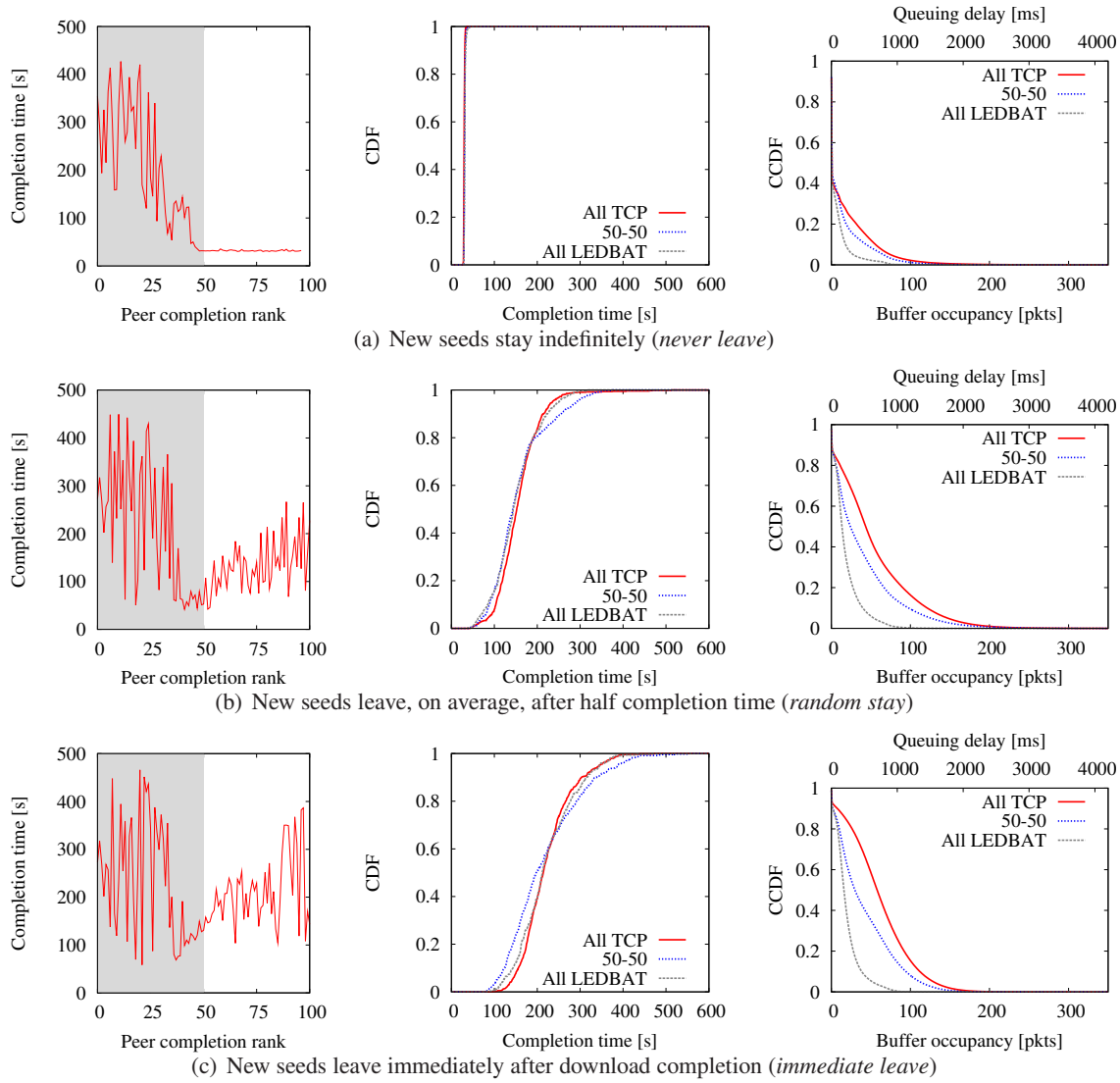


Figure 5.2: Swarm performance for different seeds holding time (never leave vs random stay vs immediate leave) and populations (homogeneous LEDBAT vs homogeneous TCP vs heterogeneous 50-50): completion time evolution and CDF, queue occupancy CCDF.

We suspect this unexpected phenomenon to arise due to (i) the decoupling of the data vs control connection, associated to (ii) the very large size of ADSL buffers: while large buffers are beneficial to backlogged data connections, they can conversely harm BitTorrent signaling. Indeed, TCP control connection competes with either LEDBAT data traffic (that strive to keep a low extra delay on the access buffer) or TCP data traffic (that indiscriminately opens up the congestion window

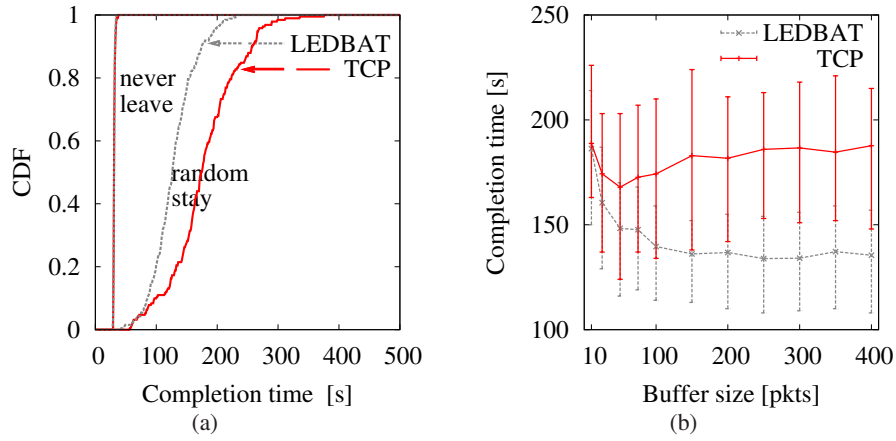


Figure 5.3: (a) Breakdown of completion time CDF according to the different peer population in the heterogeneous 50-50 scenario, for different seed holding times. (b) Swarm completion time (mean, 1st and 3rd quartiles) as a function of the buffer size B for the 50-50 random-stay scenario, for different peer population.

until loss occur). As ADSL buffers are large, queuing delay of TCP peers can grow up to seconds (Fig. 5.2): hence, this possibly hampers the performance of TCP peers, whose control traffic is significantly slowed down by competing chunk upload to other peers and by ACK traffic of downloaded chunks. Conversely the shorter queuing delay of LEDBAT peers lead to more responsive control connections, that opportunistically “steal” download slots from TCP peers (whose request rate is, as we just saw, slowed down due to self-induced congestion in the access link), as they are faster in filling the request buffer of other peers. Conversely, this phenomenon was not observed in case of homogeneous TCP swarms, as all peers fairly competed against each other.

To confirm this intuition, we carried out a set of experiments varying the buffer size of the access links. Completion time statistics (mean, 1st and 3rd quartile of the distribution) are reported in Fig. 5.3-(b) for the 50-50 random-stay scenario, with separate curves for LEDBAT and TCP peers. According to the results presented in Chap. 3, LEDBAT can be affected by the buffer size only whenever the target queuing delay exceeds the buffer size, in which case LEDBAT becomes as aggressive as TCP and becomes a loss-driven protocol. Fig. 5.3-(b) confirms this, in that whenever the buffer size cannot sustain the target (e.g., $B = 10$ ms corresponding to a 12 ms full-payload equivalent delay lower than the 100 ms target), LEDBAT and TCP performance cannot be distinguished. Conversely, LEDBAT and TCP show an opposite behavior for increasing buffer size: TCP data transfer will benefit of the increasing buffer size, translating into slower signaling rates, while LEDBAT data transfers will limit the extra queueing delay, whatever the buffer size may be (so that the completion time is roughly unaltered when $B > 50$).

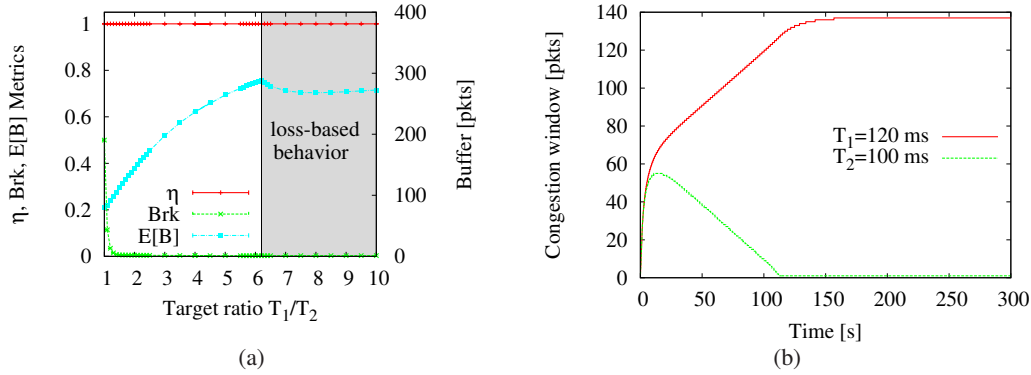


Figure 5.4: (a) Impact of target heterogeneity on performances of two LEDBAT flows and (b) time evolution of the congestion window in case $T_1 = 1.2 T_2$.

5.2.2 Target heterogeneity

After the initial assessment with fixed protocol parameters, we pursue the analysis by investigating the impact of heterogeneous target on the protocol performances, under both (i) flow and (ii) BitTorrent swarm point of view, so to assess if an actual advantage could be reached changing the default value. As far as the flow point of view is concerned, we run a set of experiment varying the target ratio T_1/T_2 in the range $[1, 10]$ where $T_2 = 100$ ms, with an approach similar to the one used in Chap. 3. Fig. 5.4-(a) reports the result of the simulations, expressed in terms of efficiency η , average queue length $E[B]$ and breakdown Brk . The first two metrics are defined in the previous Sec. 3.2.2, while the latter one is expressed as the percentage of traffic transmitted by the first LEDBAT flow, with a variable target T_1 , over the total amount of data sent over the link, i.e. $Brk = \frac{x_1}{(x_1+x_2)}$.

As we saw in the previous chapter, even a small difference in target between the two backlogged flows causes important variation in the performance. Due to the *unfairness issue* highlighted in Chap. 4, the flow with the highest target value is more aggressive, thus exploiting all the available resources to the detriment of the flow with the fixed target, as expressed by the Brk metric. The unfair situation persists also when the sum of the two target values exceed the buffer capacity, which happen when $T_1/T_2 = 6$. The gray-shaded zone in Fig. 5.4-(a) highlights the region in which both flows face packet drops, thus experiencing the TCP-like *loss-based* behavior. The time evolution of the congestion window in the case the second flow has a slightly higher target than the first one, is reported in Fig. 5.4-(b).

The second part of the analysis aims at gauging the impact of the completion time in case we consider an heterogeneous target among peers within the same swarm. In this scenario, half of the population has a fixed target $T_2 = 100$ ms, while the second half uses a different target T_1 so to explore a ratio of $T_1/T_2 \in [1, 10]$. Fig. 5.5-(a) reports the average completion time of the two set of peers, in the *random stay* scenario. As we can see, in this case the peers having a lower

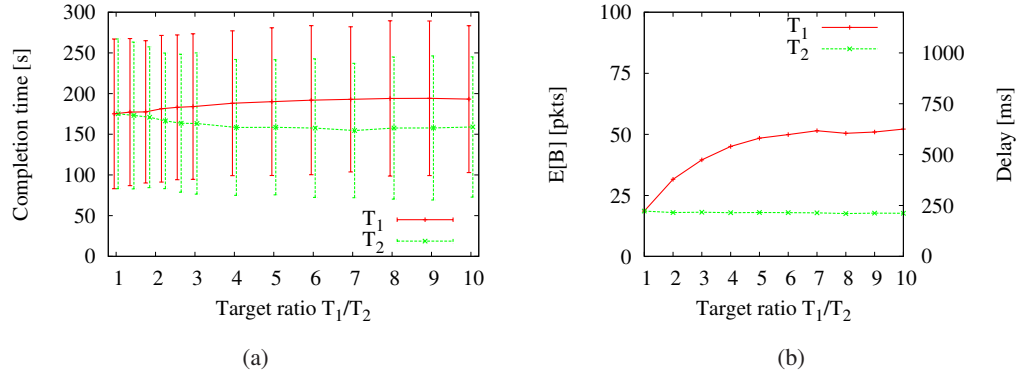


Figure 5.5: (a) Average completion time and (b) average Buffer occupancy of peers with heterogeneous target within the same swarm.

target are able to achieve smaller completion time on average, with respect to the peers with a higher value T_1 . This happens because peers with a lower target setting are more responsive in the control plane, as they strive to keep less data into the buffer. Thus, they face a lower self-induced congestions, as suggested also from the average buffer occupancy $E[B]$ reported in Fig. 5.5-(b). On the contrary, as the T_1 value increases, the congestion protocol used by those peers will become more and more aggressive, thus slowing down the timely dissemination of content and requests. The measured average delay of peers using T_1 settles to 630 ms, which is more than the double of the value measured for counterpart peers which use the smaller T_2 . These results further emphasize the subtleness of the target parameter settings, as we recap in the summary.

5.3 Summary

In this chapter we studied the impact of congestion control policies on the main BitTorrent performance metric, namely, the torrent completion time. We perform simulation in `ns2`, integrating the open-source implementations of the swarming protocol [41] and our own code for the new congestion control protocol [7]. We simulate small size swarms consisting of 100 peers, in mild flash crowd scenarios. Peers initiate data connections with either LEDBAT or TCP, but can accept any flavor of incoming connections. Swarm population is either homogeneous (i.e., all new LEDBAT or all old TCP clients) or heterogeneous (i.e., half new and half old clients). Newborn seeds either stay in the system forever, or leave immediately after completion, or after a random time, and are then replaced by a new leecher.

Our simulation results show an interesting and unexpected behavior: the *download time* at torrent completion can be moderately affected by the congestion control preferences on the *uplink traffic* direction. More precisely, in a heterogeneous peer population scenario where seeds leave after a random time or immediately, we observed that, as the use of LEDBAT practically bounds

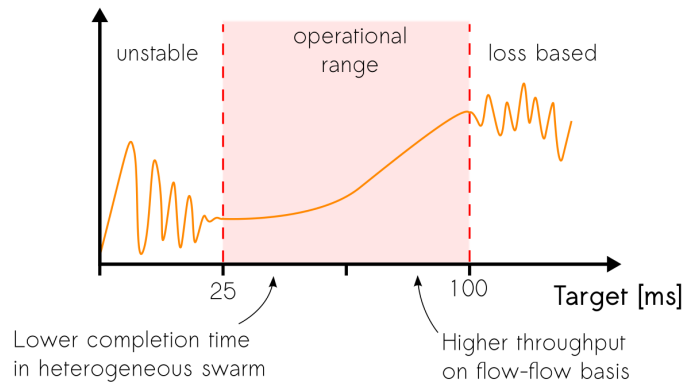


Figure 5.6: Recap of LEDBAT performances for varying target setting.

the queuing delay, LEDBAT can assist peers into faster signaling as a side effect. In turn, this translates into LEDBAT peers opportunistically gaining download slots faster than TCP peers, whose control traffic competes with a self-congested access link.

As previously highlighted in Chap. 3, the setting of the `TARGET` is rather a delicate point. The last versions of the IETF Draft [79] specification, as well as the BitTorrent BEP-29 [71], suggest a value equal to 100 ms. However different implementations with heterogeneous settings may exist, as the target was earlier specified to a smaller 25 ms. Moreover, as in the official μ Torrent client the target value can be changed, one can wonder which is the better setting to achieve optimal performances.

We summarize our findings on the target settings gathered in Chap. 3 and in this chapter with the help of the picture Fig. 5.6. From the flow point of view, a very small target leads to unstable performances, as the LEDBAT controller strives to limit the buffer occupancy to low values, for which few packets makes the difference in the induced queue delay. At the same time, as recommended in the LEDBAT Draft, values higher than 100 ms should not be used, so to avoid interfering with delay-sensitive application that can share the bottleneck link.³ Moreover, depending on the actual buffer size and bottleneck capacity, a target set too high can lead to the saturation of the buffer capacity and thus force LEDBAT flows to fall in a loss-based behavior. In this chapter we saw that from the swarm point of view, in the case we have heterogeneous target settings among peers of the same swarm, a low target value is preferable, as it leads to lower completion time. Finally, we must recall that not a single value of target, which is able to adapt to both low and high-capacity links at the same time, exists.

Overall, while we believe open-source `ns2` implementations of BitTorrent and LEDBAT to be extremely valuable for the research community, we also believe simulation not to be an entirely

³Along similar lines, ITU recommends [51] to limit the One-Way Delay to values less than 150 ms for an acceptable quality of voice communication.

faithful representation of the real world: e.g., [41] does not implement BitTorrent anti-snubbing and end-game algorithms, while our LEDBAT implementation uses a TCP framing, with a different overhead than the actual LEDBAT implementation. Moreover, we made the assumption that each peer can determine which congestion control use among TCP and LEDBAT. This however is not entirely true, as we later discover that an *hardcoded LEDBAT preference* is present in the μ Torrent client. Therefore, another direction we pursued to investigate further this phenomena is to perform real world experiments, using platform as Grid'5000, as we tackle in the following Chap. 6.

Chapter 6

Impact of LEDBAT on BitTorrent - Experimental approach

Our previous study presented in Chap. 5 suggests that LEDBAT performs better than standard TCP, as the use of LEDBAT practically limits the queuing delay to a small target, this translates into faster signaling as a side effect. However, results in the previous chapter are based solely on ns2 simulation: it becomes thus imperative to assess whether the observed phenomena also happens in practice, which is precisely the scope of the study carried out in this chapter.

We run BitTorrent applications in a flash crowd scenario over the Grid'5000 platform [5], with special attention to the main user-centric metric, the torrent completion time. Results of our experiments confirm our previous simulation results, in that, as observed in Chap. 5, LEDBAT can reduce the torrent download time. Yet, this experimental study brings new insights beyond previous findings. Currently, the default settings of BitTorrent yield to the use of a *mixture of TCP and LEDBAT traffic*. Hence, in this chapter we evaluate how this choice performs compared to the cases in which all peers use only a single protocol between TCP and LEDBAT. In this case, our experimental results show that completion time under heterogeneous swarms can be even lower than all LEDBAT (and, clearly, all TCP) swarms.

The remainder of this chapter is organized as follows. First, Sec. 6.1 reports preliminary insights on low-level BitTorrent settings gathered from a local testbed, which are instrumental to our experiments, whose results are then reported in Sec. 6.2. Finally, we summarize the findings in Sec. 6.3.

6.1 Preliminary Insights

As previously stated, the LEDBAT protocol aims at jointly (i) being efficient by fully exploiting the link capacity when no other traffic is present, and (ii) being low priority by yielding to other competing traffic on the same bottleneck. In order to achieve both these goals, LEDBAT needs to insert only a limited amount of packets in the bottleneck buffer. On the one hand, since the queue

is non empty, the capacity is fully exploited. On the other hand, as the queuing delay in the buffer remains bounded, this does not harm interactive applications.

LEDBAT exploits the ongoing data transfer to measure the One-Way Delay (OWD) on the forward path. While measuring the OWD is notoriously tricky among non-synchronized Internet hosts, LEDBAT is interested in the *difference* between the current OWD and the minimum OWD ever observed (used as an approximate reference of the base propagation delay). In turn, this OWD difference yield to a measure of the current *queuing delay*, that is used to drive the congestion window dynamics: when the measured queuing delay is below a given *target delay*, the congestion window grows, but when the queuing delay exceeds the target the congestion window shrinks.

The impact of this new protocol on the performance of BitTorrent can be affected by essentially two different settings. At a single flow level, LEDBAT is primarily driven by the LEDBAT *target delay* setting. At a swarm level, peers *relative preference* for TCP vs LEDBAT protocols plays an important role as well. Hence, before running a full-fledged set of experiments, we need to get some preliminary insights on the settings of the above two parameters. In more details, these are: (i) *net.utp_target_delay*, that tunes the value of LEDBAT delay of each flow, and (ii) *bt.transp_disposition*, that drives TCP vs LEDBAT preference of the BitTorrent client.

To this aim, we perform a battery of tests in a completely controlled environment involving one seed and two leechers, all running the latest version of the μ Torrent client available for Linux (3.0 build 25053, released on March 2011). Clients in the local testbed are interconnected by a 100 Mbps LAN and we use the Hierarchical Token Bucket (HTB) of *netem* to emulate an access bottleneck on the PC running the seed, whose uplink capacity is then capped at 5 Mbps. Any additional delay was added on the testbed links. The tracker is private within the testbed and is used to announce a set of three different torrent files having different file and chunk sizes (file size of 10, 50 and 100 MBytes, and chunk size of 256, 512 and 1024 KBytes respectively).

To understand the BitTorrent settings, we tweaked the default configuration¹ aiming at (i) verifying the compliance of *net.utp_target_delay* to the imposed delay and (ii) understanding how *bt.transp_disposition* settings, which controls when LEDBAT is used, impacts the performance of BitTorrent.

In the case of (i) *net.utp_target_delay*, usually a single experiment is sufficient to verify its compliance, since every flow obeys its own setting. Conversely multiple experiments were necessary in the case of (ii) *bt.transp_disposition*, since the behavior of a peer is affected by the *bt.transp_disposition* value of other peers as well.

In the following we summarize the most relevant findings of the local testbed experiments. Overall, in these tests we captured about 2 GBytes of packet level traces, that we make available to the scientific community at [10]. The remainder of the experiments, performed on the Grid'5000 platform, are reported in Sec. 6.2.

¹ μ Torrent clients store their configuration in a file which is not directly editable (as it also contains an hash value on the configuration content, performed by the client itself) and moreover Linux GUIs do not offer the possibility of modifying the default settings. However, the configuration file format used by Linux clients is the same as the one of the Windows clients: hence, we used Windows GUIs to produce a pool of configuration files, that we later loaded in Linux.

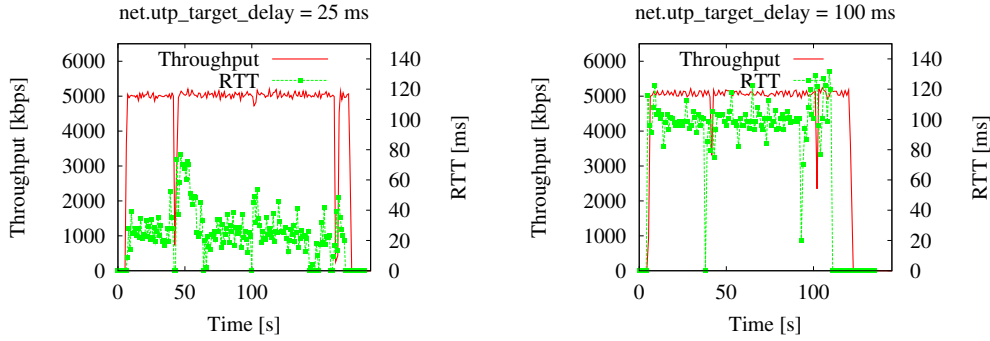


Figure 6.1: LEDBAT Target Settings: RTT and throughput for a single flow with *net.utp_target_delay*=25 ms (left) and *net.utp_target_delay*=100 ms (right).

6.1.1 *net.utp_target_delay*: Target delay settings

The *net.utp_target_delay* parameter stores the value of the LEDBAT target delay in milliseconds, and its default value is equal to 100 ms as stated in the BEP-29 [71]. Furthermore, LEDBAT is also standardized at the IETF LEDBAT Working Group [79], which specifies 100 ms as a *mandatory upper bound value* (while earlier version of the Draft referred to a 25 ms delay target).

Yet, the GUI of the Windows client allows to modify its default value, opening the way for competition between legacy applications. This behavior is confirmed by Fig. 6.1, where we show two experiments, performed at different times, where a single LEDBAT flow sends data on a 5 Mbps bottleneck, with different values of the LEDBAT target delay. We see in Fig. 6.1 that for both target delay settings, BitTorrent is using the entire available capacity, and the end-to-end delay corresponds to the target that we set by means of the *net.utp_target_delay* parameter.

As the uTP/LEDBAT specifications [71, 79] refer to a mandatory target value, we comply to the standard and focus in the remainder on the study of swarms with the same default value for *net.utp_target_delay*. At the same time, we point out that as a future work, it would be interesting to see whether, by tweaking the *net.utp_target_delay* value, some peers (or some applications) can gather an advantage over the rest of the swarm, as we saw in simulation in Sec. 5.2.2.

6.1.2 *bt.transp_disposition*: TCP vs uTP settings

The second parameter, namely *bt.transp_disposition*, controls which protocol is used for the incoming and outgoing data connection of the client. As reported in the online μ Torrent manual², *bt.transp_disposition* is a bitmask that sums up the following behaviors:

- 1: attempt outgoing TCP connections

²http://www.bittorrent.com/help/manual/appendixa0212#bt.transp_disposition

- 2: attempt outgoing LEDBAT connections
- 4: accept incoming TCP connections
- 8: accept incoming LEDBAT connections
- 16: use the new LEDBAT header format

μ Torrent default value is 31, which means that the client will accept both TCP and LEDBAT flavors, for either sending or receiving data, possibly using the new LEDBAT header format.

To understand the implications of *bt.transp_disposition* settings, we perform a number of tests with heterogeneous settings of the client. Notice that the parameter space we explore and that we make available at [10] is larger than the one reported in Tab. 6.1. Yet, for the sake of simplicity, we only report in the table the cases that we later study in Sec. 6.2, which already conveys some interesting information. Notice also that in all the experiments, the seed is set with the default value *bt.transp_disposition*=31.

In *Case 1*, the two leechers A and B have different setting for the *bt.transp_disposition* parameter: more precisely, A should attempt data connection only using TCP while B should use LEDBAT (and both will accept every flavor in reception). Our experiments show that in this scenario, peer B sends data to peer A using the LEDBAT protocol, which is the expected behavior. However, peer A sends data to peer B using the LEDBAT protocol too, which happens consistently over all repetitions, and irrespectively of file and chunks size. The reason is that when a LEDBAT connection from B to A is opened, peer A can use this opened bidirectional connection to send data to peer B. Besides, as confirmed by Arvid Norberg, one of the main BitTorrent developers, the μ Torrent client has a *hardcoded LEDBAT preference*, so that in case both a TCP and a LEDBAT connection will be successfully established, the former will be closed and only the latter will be used. As we will see in Sec. 6.2, this preference has some important (and beneficial) consequences on the overall swarm completion time.

In *Case 2*, leecher A attempts and accepts only connection via TCP (as a legacy BitTorrent like the old v1.8.2 implementation would do), while leecher B maintains the default value for *bt.transp_disposition* (which means to attempt and accept both protocols). In this scenario, any communication between the two peers are performed using the TCP protocol, which is consistent and expected for backward compatibility with older BitTorrent clients.

Other cases, not shown in Tab. 6.1, yield to different shares of traffic between TCP and LEDBAT. At the same time, since the number of leechers is small, the exact value of the breakdown is heavily influenced by the seeder flavor as well. As such, we defer a quantitative analysis of such a breakdown in the next section.

6.2 Experimental Results

We now report the experimental results on the impact of LEDBAT and TCP transport on the torrent completion time. First we briefly describe the Grid'5000 experimental platform and then focus

Table 6.1: TCP vs LEDBAT BitTorrent transport disposition.

case	peer	attempt	accept	disp	B → A	A → B	comment
1	A	TCP	*	13	LEDBAT	LEDBAT	Hardcoded LEDBAT preference
	B	LEDBAT	*	14			
2	A	TCP	TCP	5 or v1.8.2	TCP	TCP	Legacy BitTorrent implementations
	B	*	*	31			

on two case studies, namely (i) homogeneous and (ii) heterogeneous swarms, depending on the *bt.transp_disposition* settings for the leechers.

Homogeneous settings refer to scenarios where all peers have either a TCP-only preference (*bt.transp_disposition*=5, which mimics the behavior of old μ Torrent versions or legacy applications), or a LEDBAT-only preference (*bt.transp_disposition*=10, in case LEDBAT will prevail over TCP), or are able to speak both protocols (*bt.transp_disposition*=31, the current default behavior, though with an hardcoded preference for LEDBAT as we have seen in Sec. 6.1.2).

Homogeneous settings provide a useful reference, but we must consider also experiments with heterogeneous scenarios that correspond to what is observed in the Internet with clients that do not support LEDBAT at all, or that support LEDBAT but as a fallback choice rather than the default one.

We therefore investigate heterogeneous settings as well, considering scenarios with different ratios of peers using LEDBAT and TCP. More precisely, we consider the case where peers are able to accept any incoming protocol, but have different preferences for the uplink protocol (*bt.transp_disposition*=13 for TCP, and *bt.transp_disposition*=14 for LEDBAT). We consider the case where the preference splits are 50/50, 25/75, or 75/25 to mimic scenarios where TCP vs LEDBAT preferences are fairly balanced, or biased toward one of the two protocols.

Notice that, while there may be several LEDBAT available, different BitTorrent applications use different default settings (i.e., sticking to TCP preference or embracing LEDBAT) depending on the success of the new protocol (and the existence of readily available libraries for different operating systems).

6.2.1 Grid'5000 Setup

We performed experiments on a dedicated cluster of machines that run Linux as the host operating system and using the μ Torrent 3.0 client as before. Hosts of the Grid'5000 platform are interconnected by an high-speed 1 Gbps LAN, and we emulate realistic bandwidth restrictions and queueing of home gateways by using the *netem* module for the Linux kernel. As noted in [79] and experimentally confirmed in Chap. 2 and by [78], ADSL modems can buffer up to a few seconds worth of traffic: in our experiments, we set the buffer size B according to the uplink capacity C so

that $B/C = 1$ second worth of traffic.

We instrumented the Linux kernel to log the queue size Q in bytes and packets after each dequeue operation, logging also the cumulative number of packets served and dropped by the queue. During our experiments, we disabled the large segment offloading [68] which ensured that the maximum segment size of the TCP and LEDBAT packets never exceeded the maximum transfer unit (MTU). In each experiment we used the Cubic flavor of TCP, the default for Linux kernels: in reason of our previous measurement effort presented in Chap. 2, we may expect Cubic to be more aggressive with respect to the standard TCP NewReno flavor, and more similar to the default Compound TCP flavor adopted in recent versions of Windows.

We use 76 machines on the Grid'5000 platform and consider an Internet flash crowd scenario, where a single seed is initially providing all the content (a 100 MBytes file) to a number of leechers all arriving at the same time (and never leaving the system). Furthermore, each BitTorrent peer (i) experiences an ADSL access bottleneck [16] and (ii) encounters a self-induced congestion, unrelated to the cross-traffic.

As for (i), we start by considering 3 simple homogeneous capacity scenarios in which we limit the leechers and seed uplink capacity to $C = 1, 2$ and 5 Mbps with the Hierarchical Token Bucket algorithm. For the sake of simplicity, as the qualitative results do not change for different values of C , in the following we report the results for $C = 1$ Mbps. While it could be objected that Internet capacity are not homogeneous, we argue that homogeneous scenarios are needed as a first necessary step before more complex and realistic environments are emulated. Additionally, the impact of heterogeneous access capacity is a well known clustering effect [20], that we believe to fall outside of our main aim, i.e., the comparison of TCP vs LEDBAT transport, and that can be studied with a future experimental campaign.

As for (ii), we are forced to map a single peer on each host of the Grid'5000 platform, as otherwise unwanted mutual influence may take place on multiple peers running on the same hosts. Given the number of hosts $N = 76$ we can use, this constrains us on the size of the swarm we investigate. However, we prefer to take a cautious approach, and avoid to introduce the aforementioned mutual influence that could bias in an unpredictable way the results of our experiments (see a discussion on the conclusions).

We repeat the experiments three times for each settings, to smooth out stochastic variability in the experiments due to BitTorrent random decisions (e.g., chunk selection, choke, optimistic unchoking etc). Also, we make results of our campaign available to the scientific community as for the local testbed at [10].

Overall, the volume of collected data in the Grid'5000 testbed amounts to about 10 GBytes. Yet, we point out that, in reason of the large number of experiments and seeds, we were not able to store packet-level traces, but only periodic transport-layer (i.e., TCP, UDP traffic amount), application layer (i.e., tracker) and queuing logs.

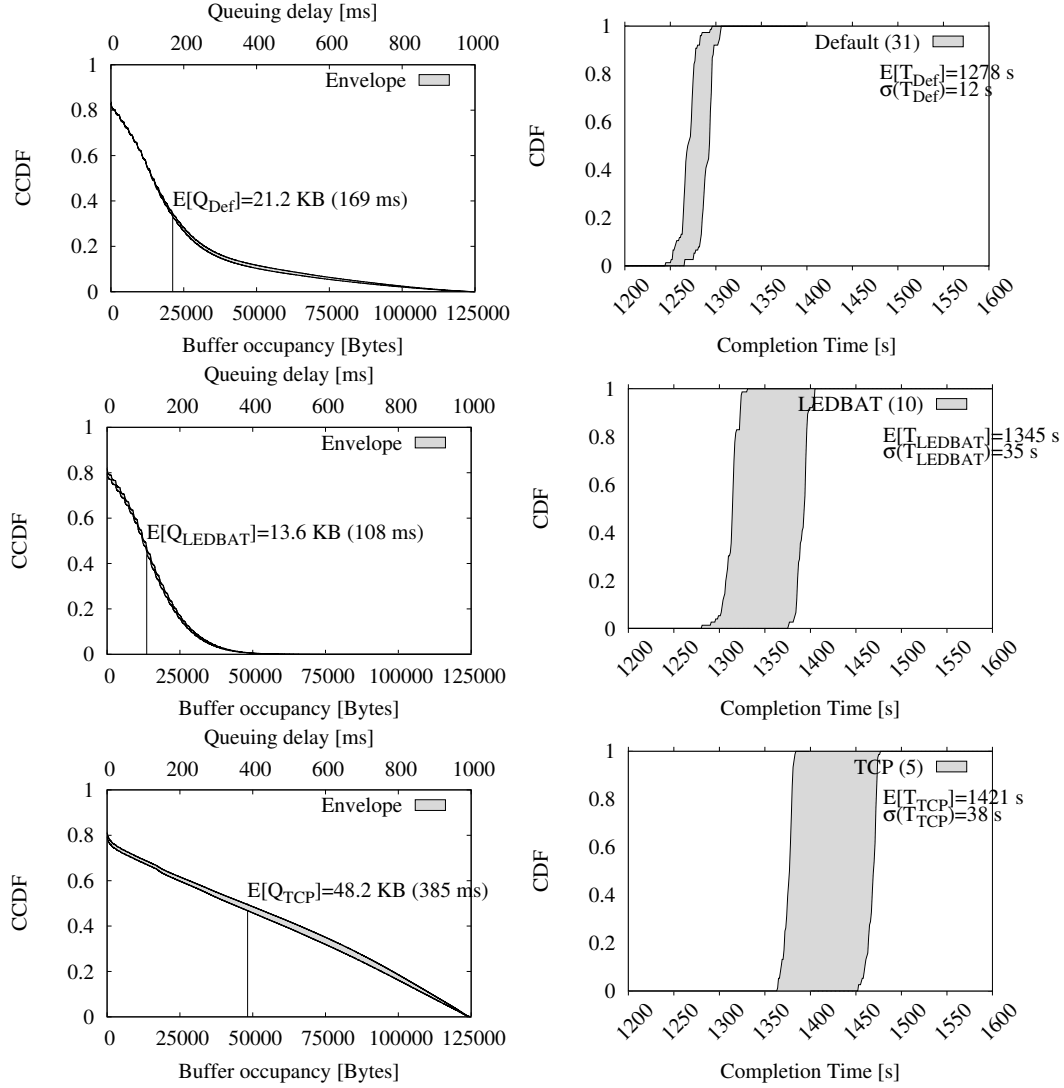


Figure 6.2: Buffer occupancy CCDF (left) and Completion time CDF (right) for homogeneous swarms: default settings (*bt.transp_disposition*=31, top), LEDBAT only swarm (*bt.transp_disposition*=10, center) and TCP only swarms (*bt.transp_disposition*=5, bottom). The vertical line in the Buffer occupancy plot represent the average of the queue length (in KB and ms).

6.2.2 Homogeneous *bt.transp_disposition* settings

Results of the homogeneous case are reported in Fig. 6.2. For each metric of interest, the figure reports the *envelope* of the gathered results, i.e., the minimum and maximum curves over the three

iterations.

We express results in terms of (i) the cumulative distribution function (CDF) of the torrent completion time T , on the right plots and of (ii) the complementary cumulative distribution function (CCDF) of the buffer occupancy Q of the access link of each peer, on the left plots. The buffer occupancy is expressed both in bytes (bottom x-axis) and in terms of the amount of delay an interactive application would experience for the emulated access capacity (top x-axis).

Additional details are reported in the inset of each figure, showing: (iii) the average $E[T]$ and standard deviation $\sigma[T]$ of the torrent completion time; (iv) the byte-wise share between TCP and LEDBAT, with a notation X/Y that specifies that $X\%$ ($Y\%$) of the bytes are carried over TCP (LEDBAT), with $X + Y = 100\%$; (v) the mean queue size $E[Q]$ in KBytes and milliseconds.

In the top row we report the scenario where all peers use default settings (*bt.transp_disposition=31*), i.e., the peers are able to speak both TCP and LEDBAT protocols. Middle plots report the case of an all-LEDBAT swarm (*bt.transp_disposition=10*), while all-TCP swarms (*bt.transp_disposition=5*) are depicted in the bottom row.

Notice that, on the long run, all swarms achieves similar efficiency: looking at the CDF of the buffer occupancy in Fig. 6.2 we can see that in roughly 80% of the time, after a dequeue operation the queue is non-empty. That efficiency is also tied to the BitTorrent system dynamics (e.g., pipelining of the requests, chunk exchange dynamics, etc.). Also the number of packets remaining in the queue after a packet transmission further depends on the congestion control protocol of choice. As expected, TCP AIMD dynamics tend to fill the buffer, while LEDBAT strives (and manages) to limit the queue size.

These behaviors translate into different completion times statistics and, especially, completion times appear to benefit from a mixture of TCP and LEDBAT traffic. We point out that, in the mixed case where BitTorrent peers are able to speak both protocols (*bt.transp_disposition=31*), the following happens: two connections, a TCP and an LEDBAT ones are attempted, and in case LEDBAT is successfully opened, it is preferred over the TCP one. This translates into a traffic mixture where about 80% of the data traffic happens to be carried over LEDBAT.

Notice that the queue size alone cannot explain the difference in the completion time statistics (as otherwise, completion time in all-LEDBAT swarms will be the lowest). Hence, we conjecture this result to be the combination of two effects –on the control and data plane– that are assisted by the use of LEDBAT and TCP respectively. First, a longer queue size due to TCP can negatively influence the completion time, by hindering a timely dissemination of *control information* (e.g., chunk interest). The longer the time needed to signal out interests, the longer the time prior to start their download, and their subsequent upload to other peers (which harms all-TCP completion time).

Notice indeed that as in the all-TCP case the one way queuing delay may reach up to 400 ms on average, this entails that RTT for signaling exchanges may be on the order of a second, that can possibly slow down significantly the chunk spreading dynamics. Consider then that BitTorrent is using pipelining to avoid a slowdown of the transfer due to the propagation delay of requests for new chunks. From our experiments, it appears that the pipelining used by BitTorrent is not large enough to deal with delays that might be encountered with xDSL connections.

However, as previously said, the completion times statistics are not fully explained in terms of the queuing delay, as otherwise all-LEDBAT swarms should be the winner. Yet, while LEDBAT limits the queue size and avoids to interfere with a timely dissemination of control messages, LEDBAT is also by design less aggressive than TCP. It follows that TCP may be more efficient for rapidly sharing *chunks in the data plane*. This can in turn harm the all-LEDBAT completion time, that is slightly larger with respect to the default settings *bt.transp_disposition=31*.

Interestingly, our previous simulation study presented in Sec. 3.2 shown that a combination of TCP and LEDBAT can increase the efficiency on the case of two flows sharing a bottleneck link. Shortly, this happens because the low-priority protocol is still able to exploit the capacity unused by TCP (as its rate increases when queuing is low), without at the same time increasing the average system queuing delay (as its rate slow down when TCP traffic increases). The experimental results of this chapter further confirm that a combination of TCP and LEDBAT can be beneficial to the completion time of the whole swarm as well. Moreover, although the exact shape of the completion time CDFs differ across experiments (due to the stochastic nature of BitTorrent chunk scheduling and peer selection decisions), the results are consistent across all iterations.

6.2.3 Heterogeneous *bt.transp_disposition* settings

Having seen that a mixture of TCP and LEDBAT protocols can be beneficial to the completion time, we further investigate different shares of TCP (*bt.transp_disposition=13*) vs LEDBAT peers (*bt.transp_disposition=14*), i.e., peers that prefer one of the two protocols for active connection open, but that can otherwise accept any incoming connections.

We consider three peer-wise shares, namely 25/75, 50/50, and 75/25 (in the X/Y notation, $X\%$ represents the percentage of peers preferring TCP on their uplink, i.e., *bt.transp_disposition=13*). These shares represent three different popularity cases of LEDBAT, that can be either the default in only few implementations of the BitTorrent applications (25/75), or compete equally (50/50) or even be dominant (75/25). We believe these shares to represent illustrative points, covering all relevant scenarios of the possible population repartition.

The plots in Fig. 6.3 additionally report (i) the average queuing delay, for all the swarm as well as for different peer classes, (ii) the peer- and byte-wise traffic shares, and (iii) the average system completion time, as well as the average completion time for peers of different classes.

As for (i), the average queuing delay statistics are as expected, with an increase of the queuing delay of LEDBAT peers due to bursty acknowledgements in reply to TCP traffic due to TCP peers in the reverse path. As for (ii), the byte-wise share closely follow the peer-wise share. Finally, let us focus on (iii) the completion time statistics. Interestingly, as Fig. 6.3 shows, while a small amount of TCP traffic is beneficial in reducing the overall swarm completion time (bottom row), a large TCP amount can instead slow down the torrent download for the whole system (top row).

Further, notice that completion times are practically the same for LEDBAT and TCP peers (with a slight advantage for the latter). Hence, differently from our previous simulation work presented in Chap. 5, we do not observe an unfairness of completion time between different peer classes within an heterogeneous swarm. This is due to the fact that the previous simulation analysis considered a

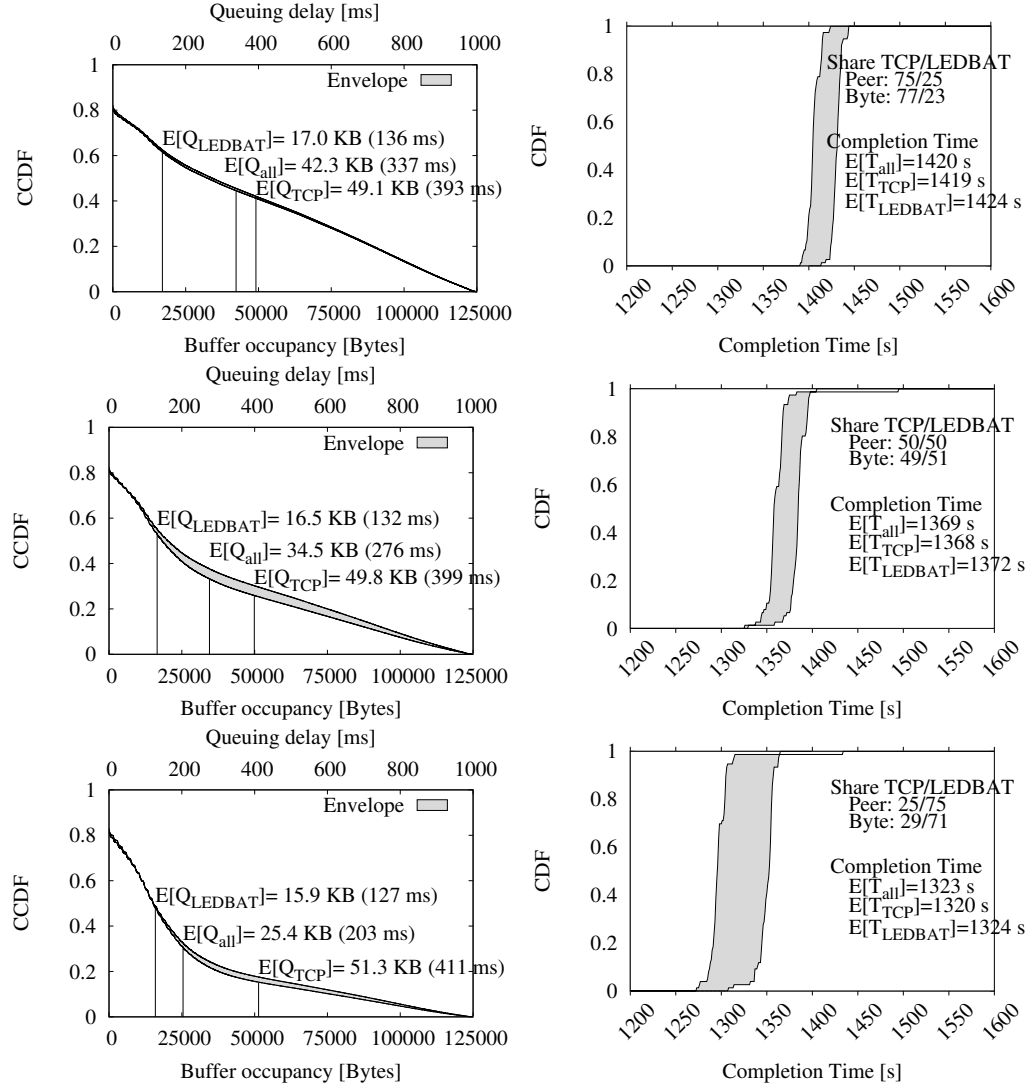


Figure 6.3: Buffer occupancy CCDF (left) and Completion time CDF (right) for heterogeneous swarms: prevalence of TCP peers (75/25, top), fair population share (50/50, middle), and prevalence of LEDBAT peers (25/75, bottom).

simpler model for *bt.transp_disposition*, that neither (i) accounted for TCP peers using an already opened LEDBAT connection in the reverse side nor (ii) for the hardcoded LEDBAT preference.

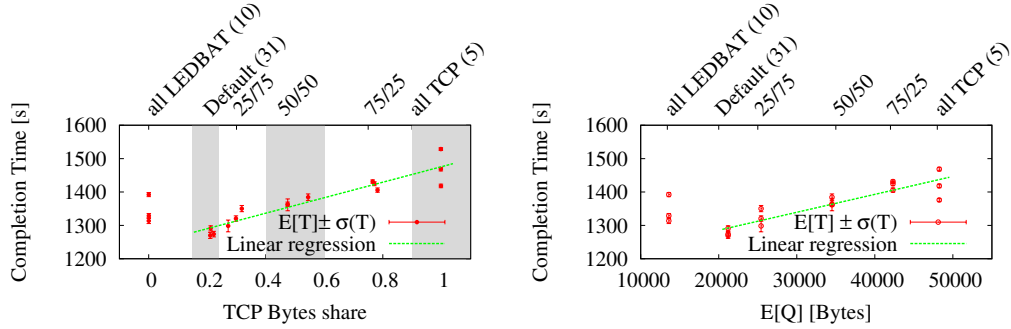


Figure 6.4: Completion time as a function of the TCP vs LEDBAT byte-wise share (left) and as a function of the average buffer occupancy (right).

6.2.4 LEDBAT vs TCP in a nutshell

Fig. 6.4 present a summary of our results considered so far. T is the completion time (mean and standard deviation) for different iterations with both homogeneous and heterogeneous swarm populations. The T metric is reported as a function of the byte-wise TCP traffic share (left plot) and of the average buffer occupancy (right plot).

Both plots also report, for TCP traffic shares different from zero (non-0 TCP) only, a linear regression of the completion time. Notice that the linear model provides a nice fit to forecast the completion time performance in presence of different TCP vs LEDBAT mixtures.

Furthermore, as observed in Sec. 5.2.1 by means of simulation, Fig. 6.4 confirms that the *completion time increases for increasing buffer occupancy*, which in turn generally increases with the amount of TCP traffic exchanged.

As previously argued, this is due to a slow-down of BitTorrent signaling traffic, while the completion time increase of all-LEDBAT swarms is instead likely due to the low-priority of LEDBAT in the data plane. Hence, we also remark a non-monotonous behavior for the completion time, that decreases for decreasing percentages of TCP traffic, and then increases again for all-LEDBAT swarms. As different dynamics takes place, hence the linear dependence only applies in case of LEDBAT and TCP traffic mixtures (i.e., non-0 TCP traffic share).

Finally, notice that the default BitTorrent settings consistently yield to the shortest download time we observed in the experiments, which confirms the soundness of the *bt.transp_disposition* design decision and settings.

6.3 Summary

In this chapter we asses the impact of LEDBAT (the new BitTorrent congestion control algorithm for data exchange) on the torrent completion time (the main user QoE metrics) by means of an

experimental campaign carried out in a fairly large scale controlled testbed.

Our results show that, in flash crowd scenarios, users will generally benefit of a mixture of TCP and LEDBAT traffic, both in homogeneous and heterogeneous swarms. Interestingly indeed, results with mixed TCP and LEDBAT traffic show consistently shorter download time with respect to the case of homogeneous swarms using either an all-TCP or an all-LEDBAT congestion control. Especially, our results confirm the soundness of default BitTorrent settings, which use both TCP and LEDBAT protocols and lead to the shortest completion time in our experiments.

This results is the combination of two effects, on the control and data plane, that are assisted by the use of LEDBAT and TCP respectively. By keeping the queue size low, LEDBAT yields to a timely dissemination of signaling information, that would otherwise incur in higher delays due to longer queues building up with TCP. At the same time, by its more aggressive behavior, TCP yields to higher efficiency in the data plane, that results in more timely dissemination of chunk content.

Chapter 7

Conclusion

7.1 Summary

In this thesis we investigate on the impact of *congestion control* protocols on distributed file-sharing application. Specifically, we concentrate our attention on the LEDBAT protocol, proposed by the BitTorrent developers for carrying the data traffic in their content distribution service. The main goal of this protocol is to implement a *Lower-than Best Effort*, which aim to realize a low priority transport service that (i) exploits the spare bandwidth, i.e., the excess bandwidth with respect to the fair rate achieved by standard TCP flows (ii) without interfering with the foreground traffic, and in particular the interactive flows.

Congestion control perspective

In the first part of the thesis we focus on the flow point of view, so to assess the transport-level performance of LEDBAT as a delay-based low-priority alternative in the wide spectrum of congestion control algorithms.

To this aim, first we use an experimental approach based on a black-box methodology to study the performances of the closed-source μ Torrent client. We exploit a local testbed to investigate the behavior of different client versions under a set of emulated artificial network conditions, as bandwidth limitations and enforced delays in the communication path. The considered scenarios tackled both single-flow as well as multiple-flows competing at the same bottleneck. Our results show that LEDBAT is a promising protocol, able to fulfill its design objectives. Specifically, it efficiently uses the spare bandwidth and limits the bufferbloat effect, adding only little to the existing queuing delay. From the tests performed over ADSL links and the ones in multiple-flow scenario, we find out that the protocol effectively yields to regular best-effort TCP flows. However, depending on the flavor and parameter settings of the competing TCP traffic, the aggressiveness of LEDBAT exhibits different behavior.

To pursue our investigation, we resort to implementing the LEDBAT protocol in the `ns2` network simulator, using the specification provided by the IETF Draft. The protocol is described as

a delay-based congestion control, which uses modifications in the One-Way Delay as indications of incipient congestion on the end-to-end path. The congestion window of the flow is adjusted by means of a linear controller in order to reach a fixed amount of inserted queuing delay, that corresponds to the `TARGET` parameter. Our simulation results confirm that this design leads to a low-priority transfer, that yields to other concurrent TCP flows. Then, we compare the performances of LEDBAT with other protocols similar in spirit, such as TCP-LP and NICE. Our results show that TCP-LP is the most aggressive of the three due to its loss-based design, while LEDBAT achieve the lowest priority among all. A sensitivity analysis on the `TARGET` delay and `GAIN` parameter of LEDBAT highlight the difficulty in tuning its level of aggressiveness with respect to TCP flows. Overall, if on the one hand the `GAIN` has almost no influence, the impact of `TARGET` is rather difficult to control, as even small changes in its value entail steep variations in performances.

The study of two, or more, LEDBAT flows that start at different times and compete at the same bottleneck revealed an *unfairness* issue, that holds both in simulations and real-world experiments on controlled testbed. This unfortunate situation makes the latecomer flow exploiting all the available capacity, at the expense of the existing transfers, which are forced to starvation. Since LEDBAT can be used for any background traffic, the normalization of a flawed protocol can have potentially dramatic effects, because it can severely impact the performance of long lived transfers, as for instance backups of home users towards virtual storage services. Our investigation identified the additive decrease component of the LEDBAT as the root cause of this issue, as already pointed out by related work on congestion window adjustment [32]. For this reason, we present fLEDBAT (“fair-LEDBAT”), a protocol modification that solves the main theoretical drawback of LEDBAT yet preserving its original goals. Our solution is proved to converge to a fair and efficient work point by means of analytical techniques and to perform under a wide range of settings through an extensive set of simulations.

BitTorrent swarm perspective

In the second part of this thesis, we focus our attention on the impact of the LEDBAT protocol on the BitTorrent performance, specifically on the torrent completion time, the metric that best define the overall users’ quality of experience.

First we resort to simulation, using our LEDBAT `ns2` implementation with the already existing BitTorrent one [2, 41]. In order to grasp the behavior of the torrent swarms with different degrees of the new protocol’s diffusion, we considered several population shares, with both homogeneous (all peers use the same transport protocol) and heterogeneous (mixed peers using TCP and LEDBAT) settings. Moreover we consider three different seed strategies, with peers that leaves immediately after the download is completed, or leave after a random time, or stay forever into the system. Our simulation results show that in a heterogeneous peer population scenario, when peers stay for a random time or leave immediately, the use of LEDBAT as uplink protocol resorts in a lower completion time. This is due to the fact that LEDBAT limits the queuing delay at the access, which in turns helps to achieve a more timely signalling and take advantage over self-congested TCP peers. To evaluate the impact of the target parameter from the swarm point of view, we setup a scenario in which peers have heterogeneous target settings, and our results show that peers with a

lower target value achieved a lower completion time.

Along similar lines, we then evaluated the impact of LEDBAT on BitTorrent using an experimental approach over the Grid'5000 [5] cluster platform. We tuned the μ Torrent client for Linux in order to set the desired target setting and the preferred transport protocol for data transmission. As in the study performed via simulations, the scenarios consider both homogeneous and heterogeneous swarm population. Results confirmed the findings gathered earlier, namely LEDBAT can assist in reducing the torrent completion time. Moreover, we discover that even better performances can be achieved under a precise mixture of TCP and LEDBAT traffic, as in the case of the default configuration of the BitTorrent client. In this case, the peer which takes advantage from both the fast signalling provided by the LEDBAT protocol and the high efficiency in the data plane given by TCP to attain better performances.

Overall, the main contributions of this thesis constitute a step toward the understanding of LEDBAT as LBE protocol and of the congestion control applied to the P2P content distribution applications in general. First, we identified a critical intra-protocol unfairness issue of the original LEDBAT algorithm and we proposed a modified solution, which is able to solve the problem without affecting the attainment of its original goals. Moreover we evaluated the impact of the LEDBAT protocol on the BitTorrent performances, discovering that it can be beneficial in improving the quality of experience of end users in terms of both enhanced interactivity –thanks to the limited queuing delay induced– and shorter content download time.

7.2 Future work

Even though we did the best we could to extensively cover our research topic, some problems and issues remains unsolved, as usual in the research field. In this section we present the open points that we would like to explore as future work direction.

Fixed target value

In our investigation on LEDBAT carried out in Chap. 4 we discover and solve a major problem of the algorithm, which is the latecomer unfairness. However, another critical point remains unsolved in our opinion, and is related to the *fixed* value for the target queuing delay. This issue, which has been pointed out also in [78] and discussed in the IETF WG of LEDBAT (for which fixed settings are seen as “magic numbers”) may lead to non compliant implementations and highlight the limitation of LEDBAT in following future network evolutions. Hence, an *adaptive* scheme will be preferable, to guarantee the lowest level of intrusiveness with respect to delay-sensitive applications, and to scale to different degree of network capacity.

Interaction between AQM and CC

We mention in the introduction about the existence of two orthogonal solutions to limit the excessive queuing on the Internet as a results of the bufferbloat phenomenon: local Active Queue

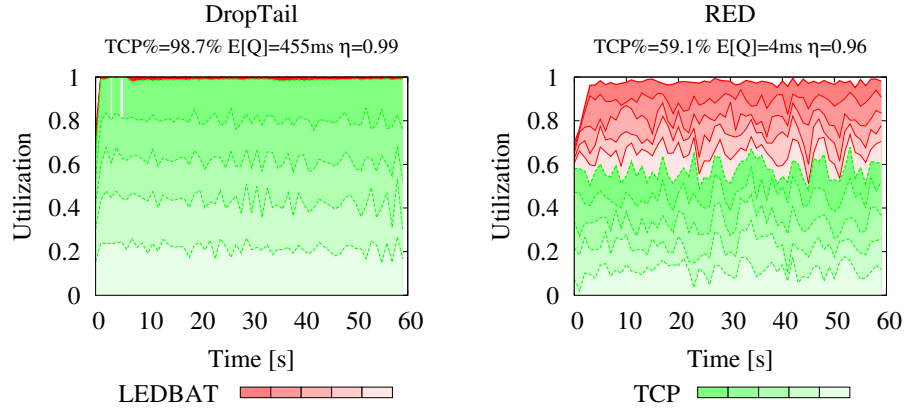


Figure 7.1: Example of problems arising from the interaction of AQM and CC techniques (ns2 simulation for RED and LEDBAT).

Management (AQM) and end-to-end Congestion Control (CC). If LEDBAT is found to be an effective way to deal with (low) prioritization and to avoid self-congestion on the access link, for the time being it is only used by the BitTorrent application. At the same time, an AQM could solve the problem for all the applications, as all the traffic of an end-user pass through the home gateway. At this point, a natural question that arise is about the interaction of these proposals. Early investigation carried out with Gong YiXi, supervised together with Dario Rossi during my last year of the thesis, showed that their interplay can have negative consequences, as AQM can induce a *reprioritization* of the congestion control protocols. In other words, current scavenging protocols can successfully realize a LBE priority only if the bottleneck buffer operates according a Drop-Tail discipline. Fig. 7.1 illustrates the phenomenon, with a breakdown of the link utilization when 5 TCP NewReno (represented with a different shade of green) and 5 LEDBAT (red) backlogged flows share the same bottleneck. Capacity is set to 10 Mbps and the buffer has room for 600 ms worth of queuing delay. Further statistics about the average queue in ms $E[Q]$, the capacity share exploited by the aggregate TCP%, and the average link utilization η are reported on top of the plots. In the DropTail case, the LEDBAT protocol operates in a LBE mode, by using the spare capacity left by NewReno, but the queuing delay approaches half a second on average. In the RED case, the queue delay is successfully limited to only 4 ms, but at the expense of a slight 3% reduction of link utilization and a complete reset of the relative level of priority between flows. We believe that more attention in this direction will be needed in the future.

Data-plane efficiency vs. Control-plane timeliness

In Chap. 6 we investigated on the impact of LEDBAT in BitTorrent performance in a controlled testbed, and discovered a connection between the data-plane efficiency and the control-plane timeliness of content dissemination. A future work in this direction will be a more careful investigation

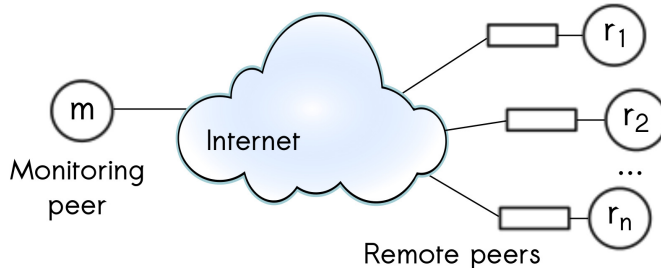


Figure 7.2: Schema of the monitoring architecture.

on the control and data plane interdependence, in an effort to assess if a content distribution system should be designed for data plane efficiency, or for control plane timeliness.

LEDBAT as a tool for inferring bufferbloat of remote hosts

In the introduction of this thesis we saw that nowadays about half of the BitTorrent data traffic is carried over LEDBAT. In order to quantify to which extent this protocol is able to reduce the queuing delay of remote hosts, we could setup a monitoring architecture, as depicted in Fig. 7.2. The architecture is based on passive analysis of BitTorrent traffic running at a local monitor peer m . This peer participates in a torrent swarm as a regular leecher and exchanges with remote peers r_i data over either LEDBAT (when available) or TCP (legacy clients). The amount of buffering delay of remote peers can be then inferred exploiting the OWD information provided by LEDBAT or, with a similar approach, the TCP timestamp option information as specified in the RFC 1323 [54]. Preliminary work, carried out by Chiara Chirichella under mine and Dario Rossi guidance, validated the methodology on a local testbed against different ground truth and set up the basis for a wider measurement campaign on the Internet.

Appendix A

List of publications

We report here the list of already published papers and submitted ones, which are related to this manuscript.

Published

- [P1] D. Rossi, C. Testa, S. Valenti, Yes, we LEDBAT: Playing with the new BitTorrent congestion control algorithm, In *Passive and Active Measurement (PAM) 2010*, Zurich, Switzerland, April 2010
 - [P2] D. Rossi, C. Testa, S. Valenti and L. Muscariello, LEDBAT: the new BitTorrent congestion control protocol, In *IEEE International Conference on Computer Communication Networks (ICCCN'10)*, Zurich, Switzerland, August 2010
 - [P3] G. Carofiglio, L. Muscariello, D. Rossi, C. Testa, A hands-on Assessment of Transport Protocols with Lower than Best Effort Priority, In *IEEE International Conference on Local Computer Networks (LCN 2010)*, Denver, Colorado, USA, October 2010
 - [P4] C. Testa, D. Rossi, On the impact of uTP on BitTorrent completion time, In *IEEE International Conference on Peer-to-Peer Computing (P2P 2011)*, Kyoto, Japan, September 2011
 - [P5] C. Testa, D. Rossi, A. Rao, A. Legout, Experimental Assessment of BitTorrent Completion Time in Heterogeneous TCP/uTP swarms, In *Traffic Measurement and Analysis (TMA 2012) Workshop at PAM 2012*, Vienna, Austria, March 2012
 - [P6] C. Chirichella, D. Rossi, C. Testa, T. Friedman, A. Pescapè', Inferring the buffering delay of remote BitTorrent peers under LEDBAT vs TCP, In *IEEE International Conference on Peer-to-Peer Computing (P2P 2012), Demo Session*, Terragona, Spain, September 2012
-

Submitted

- [S1] G. Carofiglio, L. Muscariello, D. Rossi, C. Testa, S. Valenti, Rethinking the Low Extra Delay Background Transport (LEDBAT) protocol, *SUBMITTED to Elsevier Computer Networks*
 - [S2] C. Chirichella, D. Rossi, C. Testa, T. Friedman, A. Pescapé, Remotely gauging upstream bufferbloat delays, *SUBMITTED to Passive and Active Measurement (PAM) conference*
 - [S3] Y. Gong, D. Rossi, C. Testa, S. Valenti, M. D. Tath, Fighting the bufferbloat: on the coexistence of AQM and low priority congestion control, *SUBMITTED to Traffic Measurement and Analysis (TMA) Workshop at PAM conference*
 - [S4] C. Testa, D. Rossi, A. Rao, A. Legout, On data plane efficiency vs control plane timeliness and P2P performance, *SUBMITTED to Traffic Measurement and Analysis (TMA) Workshop at PAM conference*
-

Bibliography

- [1] Apple LEDBAT congestion control module for Mac OS X. http://opensource.apple.com/source/xnu/xnu-1699.22.81/bsd/netinet/tcp_ledbat.c.
 - [2] BitTorrent in ns2. <https://sites.google.com/site/koljaeger/bittorrent-simulation-in-ns-2>.
 - [3] BitTorrent Specification Wiki. <http://wiki.theory.org/BitTorrentSpecification>.
 - [4] Comcast throttles bittorrent traffic, seeding impossible. <http://torrentfreak.com/comcast-throttles-bittorrent-traffic-seeding-impossible/>.
 - [5] Grid'5000 platform. <https://www.grid5000.fr>.
 - [6] LEDBAT Mailing List Archives. <http://www.ietf.org/mail-archive/web/ledbat>.
 - [7] LEDBAT ns2 code. <http://perso.telecom-paristech.fr/~drossi/index.php?n=Software.LEDBAT>.
 - [8] LEDBAT Working Group (WG) Charter. <http://datatracker.ietf.org/wg/ledbat/charter/>.
 - [9] libtorrent project - Rasterbar. <http://www.rasterbar.com/products/libtorrent>.
 - [10] Testbed traces archive. <http://perso.telecom-paristech.fr/~testa/pmwiki/pmwiki.php?n=Main.BTtestbed>.
 - [11] The Network Simulator ns2. <http://www.isi.edu/nsnam/ns/>.
 - [12] Microsoft background intelligent transfer service (bits). <http://msdn.microsoft.com/en-us/library/aa363167.aspx>, Oct 2001.
 - [13] BitTorrent Calls UDP Report "Utter Nonsense". <http://tech.slashdot.org/article.pl?sid=08/12/01/2331257>, Dec 2008.
-

- [14] A. Abu. Performance Evaluation and Improvement of the Low Extra Delay Background Transport Congestion Control Algorithm. <https://sites.google.com/site/amudajames/publications>.
 - [15] A. Abu and S. Gordon. A Dynamic Algorithm for Stabilising LEDBAT Congestion Window. In *2nd IEEE International Conference on Computer and Network Technology (ICCNT 2010)*, Bangkok, Thailand, Apr 2010.
 - [16] A. Akella, S. Seshan, and A. Shaikh. An empirical evaluation of wide-area internet bottlenecks. In *3rd ACM SIGCOMM Conference on Internet Measurement (IMC'03)*, Miami, FL, USA, Oct 2003.
 - [17] S. Alcock and R. Nelson. Application flow control in YouTube video streams. *ACM SIGCOMM Comp.Comm. Rev.*, 41(2):24–30, 2011.
 - [18] E. Alessandria, M. Gallo, E. Leonardi, M. Mellia, and M. Meo. P2P-TV Systems under Adverse Network Conditions: A Measurement Study. In *28th IEEE Conference on Computer Communications (INFOCOM 2009)*, Rio de Janeiro, Brazil, Apr 2009.
 - [19] M. Andreica, N. Tapus, and J. Pouwelse. Performance Evaluation of a Python Implementation of the New LEDBAT Congestion Control Algorithm. In *17th IEEE International Conference on Automation Quality and Testing Robotics (AQTR 2010)*, Cluj-Napoca, Romania, May 2010.
 - [20] Arnaud Legout and Nikitas Liogkas and Eddie Kohler and Lixia Zhang. Clustering and sharing incentives in bittorrent systems. In *Proc. of ACM SIGMETRICS'07*, San Diego, CA, USA, Jun 2007.
 - [21] R. Bennett. The next Internet meltdown. http://www.theregister.co.uk/2008/12/01/richard_bennett_utorrent_udp/, Dec 2008.
 - [22] A. R. Bharambe, C. Herley, and V. N. Padmanabhan. Analyzing and Improving a BitTorrent Performance Mechanisms. In *25th IEEE Conference on Computer Communications (INFOCOM 2006)*, Barcelona, Spain, Apr 2006.
 - [23] R. Bindal, P. Cao, W. Chan, J. Medved, G. Suwala, T. Bates, and A. Zhang. Improving traffic locality in BitTorrent via biased neighbor selection. In *26th IEEE International Conference on Distributed Computing Systems (ICDCS 2006)*, Lisboa, Portugal, Jul 2006.
 - [24] T. Bonald, M. May, and J. Bolot. Analytic evaluation of RED performance. In *19th IEEE Conference on Computer Communications (INFOCOM 2000)*, Tel Aviv, Israel, Mar 2000.
 - [25] D. Bonfiglio, M. Mellia, M. Meo, and D. Rossi. Detailed Analysis of Skype Traffic. *IEEE Transaction on Multimedia*, 11(1):117–127, Jan 2009.
-

- [26] C. Boutremans and L. B. Jean-Yves. A note on the Fairness of TCP Vegas. In *International Zurich Seminar on Broadband Communications*, 2000.
 - [27] L. Brakmo, S. O'Malley, and L. Peterson. TCP Vegas: new techniques for congestion detection and avoidance. *ACM SIGCOMM Comp. Comm. Rev.*, 24(4):24–35, 1994.
 - [28] G. Carofiglio, L. Muscariello, D. Rossi, and S. Valenti. The quest for LEDBAT fairness. In *14th IEEE Global Communication (GLOBECOM 2010)*, Miami, FL, USA, Dec 2010.
 - [29] M. W. D. Center. TCP Receive Window Size and Window Scaling. <http://msdn.microsoft.com/en-us/library/ms819736.aspx>.
 - [30] Cerf, V. and Jacobson, V. and Weaver, N. and Gettys, J. BufferBloat: what's wrong with the Internet? *Commun. ACM*, 55(2):40–47, Feb 2012.
 - [31] Cheshire, S. It's the latency, stupid! <http://rescomp.stanford.edu/~cheshire/rants/Latency.html>, May 1996.
 - [32] D. Chiu and R. Jain. Analysis of the increase and decrease algorithms for congestion avoidance in computer networks. *Computer Networks and ISDN systems*, 17(1):1–14, 1989.
 - [33] M. Christiansen, K. Jeffay, D. Ott, and F. Smith. Tuning RED for Web traffic. In *ACM SIGCOMM Comm. Comp. Rev.*, volume 30, pages 139–150, 2000.
 - [34] Cisco System. Cisco Visual Networking Index: Forecast and Methodology, 2011-2016. http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11\discretionary-481360_ns827_Networking_Solutions_White_Paper.html, May 2012.
 - [35] D. Ciullo, M. Mellia, M. Meo, and E. Leonardi. Understanding P2P-TV systems through real measurements. In *IEEE Global Telecommunications Conference (GLOBECOM'08)*, pages 1–6, New Orleans, LA, USA, Nov 2008.
 - [36] B. Cohen. How has BitTorrent as a protocol evolved over time. <http://www.quora.com/BitTorrent-protocol-company>.
 - [37] B. Cohen. Incentives build robustness in BitTorrent. In *Workshop on Economics of Peer-to-Peer systems*, Cambridge, MA, USA, Jun 2003.
 - [38] B. Cohen and A. Norberg. Correcting for clock drift in uTP and LEDBAT. In *Invited talk at 9th USENIX International Workshop on Peer-to-Peer Systems (IPTPS 2010)*, San Jose, CA, USA, Apr 2010.
 - [39] L. De Cicco, S. Mascolo, and V. Palmisano. Skype video responsiveness to bandwidth variations. Braunschweig, Germany, May 2008.
-

- [40] M. Dischinger, A. Haeberlen, K. Gummadi, and S. Saroiu. Characterizing residential broadband networks. In *7th ACM SIGCOMM Conference on Internet Measurement (IMC'07)*, San Diego, CA, USA, Oct 2007.
 - [41] K. Eger, T. Hoßfeld, A. Binzenhofer, and G. Kunzmann. Efficient simulation of large-scale p2p networks: packet-level vs. flow-level simulations. In *ACM UPGRADE-CN*, Monterey, CA, USA, Jun 2007.
 - [42] Finamore, A. and Mellia, M. and Meo, M. and Munafo, M. and Rossi, D. Experiences of Internet Traffic Monitoring with Tstat. *IEEE Network Magazine, Special Issue on Network Traffic Monitoring and Analysis*, May 2011.
 - [43] S. Floyd and T. Henderson. RFC 2582: The NewReno Modification to TCP's Fast Recovery Algorithm. RFC 2582, Apr 1999.
 - [44] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, 1993.
 - [45] Franichi, D. Megaupload (and Megavideo) shut down by the Feds. <http://popwatch.ew.com/2012/01/19/megaupload-megavideo-shut-down-fbi/>, Jan 2012.
 - [46] Q. Gao and Q. Yin. Adaptive Vegas: A Solution of Unfairness Problem for TCP Vegas. In C. Kim, editor, *Information Networking. Convergence in Broadband and Mobile Networking*, volume 3391 of *Lecture Notes in Computer Science*, pages 132–141. Springer Berlin / Heidelberg, 2005.
 - [47] Garcia-Dorado, J. and Finamore, A. and Mellia, M. and Meo, M. and Munafò, M. Characterization of ISP Traffic: Trends, User Habits, and Access Technology Impact. *IEEE Transactions on Network and Service Management*, 9(2):1–14, Feb 2012.
 - [48] G. Hasegawa, M. Murata, and H. Miyahara. Fairness and stability of congestion control mechanisms of TCP. *Telecommunication Systems*, 15:167–184, 2000.
 - [49] G. Hazel. uTorrent Transport Protocol library. <http://github.com/bittorrent/libutp>.
 - [50] S. Hemminger et al. Network emulation with NetEm. In *Linux Conf Australia (LCA 2005)*, Canberra, Australia, Apr 2005.
 - [51] G. ITU. One-way transmission time. *International Telecommunication Union*, 2, 2003.
 - [52] M. Izal, G. Urvoy-Keller, E. Biersack, P. Felber, A. Al Hamra, and L. Garces-Erice. Dissecting bittorrent: Five months in a torrent's lifetime. In *Passive and Active Measurement (PAM 2004)*, Antibes, France, Apr 2004.
 - [53] V. Jacobson. Congestion avoidance and control. Aug 1988.
-

- [54] V. Jacobson, R. Braden, and D. Borman. RFC 1323: TCP Extensions for High Performance. RFC 1323, May 1992.
 - [55] R. Jain. A delay-based approach for congestion avoidance in interconnected heterogeneous computer networks. *ACM SIGCOMM Comp. Comm. Rev.*, 19(5):56–71, 1989.
 - [56] R. Jain and K. Ramakrishnan. Congestion avoidance in computer networks with a connectionless network layer: concepts, goals and methodology. Apr 1988.
 - [57] P. Key, L. Massoulié, and B. Wang. Emulating low-priority transport at the application layer: a background transfer service. In *ACM SIGMETRICS*, New York City, NY, USA, Jun 2004.
 - [58] Kreibich, C. and Weaver, N. and Nechaev, B. and Paxson, V. Netalyzr: Illuminating the edge network. In *ACM Internet Measurement Conference (IMC 2010)*, Melbourne, Australia, Nov 2010.
 - [59] M. Kühlewind and S. Fisches. Evaluation of different decrease schemes for LEDBAT congestion control. *Energy-Aware Communications*, pages 112–123, 2011.
 - [60] A. Kuzmanovic and E. Knightly. TCP-LP: A distributed algorithm for low priority data transfer. In *22th IEEE Conference on Computer Communications (INFOCOM 2003)*, San Francisco, CA, USA, Mar 2003.
 - [61] D. Leith, R. Shorten, G. McCullagh, L. Dunn, and F. Baker. Making Available Base-RTT for Use in Congestion Control Applications. *IEEE Communications Letters*, 12:429–431, Jun 2008.
 - [62] S. Liu, T. Basar, and R. Srikant. TCP-Illinois: A loss-and delay-based congestion control algorithm for high-speed networks. *ACM Performance Evaluation*, 65(6-7):417–440, 2008.
 - [63] S. Liu, M. Vojnovic, and D. Gunawardena. 4CP: Competitive and Considerate Congestion Control Protocol. In *ACM SIGCOMM*, Pisa, Italy, Sep 2006.
 - [64] P. Marciniak, N. Liogkas, A. Legout, and E. Kohler. Small is not always beautiful. In *7th International Conference on Peer-to-peer Systems (IPTPS 2008)*, Tampa Bay, FL, USA, Feb 2008.
 - [65] P. McKenney. Stochastic fairness queueing. In *9th IEEE Conference on Computer Communications (INFOCOM 1990)*, San Francisco, CA, USA, Jun 1990.
 - [66] Mellia, M. and Meo, M. and Muscariello, L. and Rossi, D. Passive analysis of TCP anomalies. *Elsevier Computer Networks*, 52(14), Oct 2008.
 - [67] J. Mo, R. J. La, V. Anantharam, and J. Walrand. Analysis and Comparison of TCP Reno and Vegas. In *18th IEEE Conference on Computer Communications (INFOCOM 1999)*, New York, NY, USA, Mar 1999.
-

- [68] Mogul, Jeffrey C. TCP offload is a dumb idea whose time has come. In *9th conference on Hot Topics in Operating Systems*, Berkeley, CA, USA, 2003.
 - [69] S. Morris. μ Torrent release 1.9 alpha 13485. <http://forum.utorrent.com/viewtopic.php?pid=379206#p379206>, Dec 2008.
 - [70] K. Nichols and V. Jacobson. Controlling queue delay. *Commun. ACM*, 55(7):42–50, Jul 2012.
 - [71] A. Norberg. BitTorrent Enhancement Proposals on uTorrent transport protocol. http://www.bittorrent.org/beps/bep_0029.html.
 - [72] A. Norberg. Dev. comment on uTP main purpose. <http://forum.utorrent.com/viewtopic.php?pid=488896#p488896>.
 - [73] R. Pan, B. Prabhakar, and K. Psounis. Choke - A stateless active queue management scheme for approximating fair bandwidth allocation. In *19th IEEE Conference on Computer Communications (INFOCOM 2000)*, Tel Aviv, Israel, Mar 2000.
 - [74] J. Pouwelse, P. Garbacki, D. Epema, and H. Sips. The BitTorrent p2p file-sharing system: Measurements and analysis. Ithaca, NY, USA, Feb 2005.
 - [75] D. Qiu and R. Srikant. Modeling and performance analysis of BitTorrent-like peer-to-peer networks. *ACM SIGCOMM Comp. Comm. Rev.*, 34(4):367–378, 2004.
 - [76] Rao, A. and Legout, A. and Dabbous, W. Can realistic bittorrent experiments be performed on clusters? In *10th IEEE International Conference on Peer-to-Peer Computing (P2P)*, Aug 2010.
 - [77] I. S. Ha, Rhee and L. Xu. CUBIC: A new TCP-friendly high-speed TCP variant. *ACM SIGOPS Operating System Review*, 42(5):64–74, 2008.
 - [78] J. Schneider, J. Wagner, R. Winter, and H. Kolbe. Out of my Way – Evaluating Low Extra Delay Background Transport in an ADSL Access Network. In *22nd International Teletraffic Congress (ITC 2010)*, Amsterdam, The Netherlands, Sep 2010.
 - [79] S. Shalunov. Low Extra Delay Background Transport (LEDBAT). IETF Draft, Oct 2009.
 - [80] K. Tan, J. Song, Q. Zhang, and M. Sridharan. A compound TCP approach for high-speed and long distance networks. In *25th IEEE Conference on Computer Communications (INFOCOM 2006)*, Barcelona, Spain, Apr 2006.
 - [81] A. Venkataramani, R. Kokku, and M. Dahlin. TCP Nice: A mechanism for background transfers. In *8th USENIX Symposium on Operating Systems Design and Implementation (OSDI'02)*, Boston, MA, USA, Dec 2002.
-

- [82] Verhulst, F. *Nonlinear differential equations and dynamical systems*. Springer-Verlag New York, Inc., New York, NY, USA, 1990.
 - [83] Wong, B. and Slivkins, A. and Sirer, E.G. Meridian: A lightweight network location service without virtual coordinates. *ACM SIGCOMM Comp. Comm. Rev.*, 35(4):96, 2005.
 - [84] Y. Xia, D. Harrison, S. Kalyanaraman, K. Ramachandran, and A. Venkatesan. Accumulation-based congestion control. *IEEE/ACM Trans. Netw.*, 13:69–80, Feb 2005.
 - [85] L. Yaning, W. Hongbo, L. Yu, C. Shiduan, and G. Simon. Friendly P2P: Application-Level Congestion Control for Peer-to-Peer Applications. In *IEEE Global Telecommunications Conference (GLOBECOM'08)*, pages 1–5, New Orleans, LA, USA, Nov 2008.
 - [86] C. Zhang, P. Dhungel, D. Wu, and K. Ross. Unraveling the BitTorrent Ecosystem. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 22(7):1164–1177, 2011.
-